A

PATENT APPLICATION
Attorney's Do. No. 6954-2

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

jc525 U.S. PTO 11/10/99

jc511 U.S. PTO 09/438084 11/10/99

EXPRESS MAIL MAILING LABEL NO. EL253055094US
DATE OF DEPOSIT: NOVEMBER 10, 1999

I HEREBY CERTIFY THAT THIS PAPER AND ENCLOSURES AND/OR FEE ARE BEING DEPOSITED WITH THE UNITED STATES POSTAL SERVICE "EXPRESS MAIL POST OFFICE TO ADDRESSEE" SERVICE UNDER 37 CFR 1.10 ON THE DATE INDICATED ABOVE AND IS ADDRESSED TO: BOX PATENT APPLICATION, ASSISTANT COMMISSIONER FOR PATENTS, WASHINGTON D.C. 20231.

TABITHA JOHNSON
(SENDER'S PRINTED NAME)

Tabitha Johnson
(SIGNATURE)

Box Patent Application
Assistant Commissioner for Patents
Washington, D.C. 20231

Enclosed for filing is a patent application under 37 CFR 1.53(b) of:

Inventors: Gregg Evan Rothermel, Margaret Myers Burnett and Lixin Li
For: METHODOLOGY FOR TESTING SPREADSHEETS

[If continuing application] This application is a [ ] continuation, [ ] divisional, [ ] continuation-in-part of prior application Serial No. _____.

Enclosures:
[X] Specification (pages 1-32); claims (pages 33-41); bibliography (pages 42-46); abstract (page 47)
[X] 15 sheet(s) of drawings
[X] Declaration or Combined Declaration and Power of Attorney
- [X] Newly executed (original or copy)
- [ ] Copy from a prior application (37 CFR 1.63(d))
- [ ] Incorporation by Reference--The entire disclosure of the prior application, from which a copy of the oath or declaration is supplied is considered as being part of the disclosure of the accompanying application and is hereby incorporated by reference therein.
- [ ] Deletion of Inventors (signed statement attached deleting inventor(s) named in the prior application (37 CFR 1.63(d)(2) and 1.33(b)

[X] Power of Attorney
[X] Assignment with cover sheet
[ ] Certified copy of priority document:

[ ] Information Disclosure Statement with Form PTO 1449
[ ] Copies of references listed on attached Form PTO-1449
[X] Declaration(s) Claiming Small Entity Status
  [X] Independent Inventor (37 CFR 1.9(f) and 1.27(b))
  [ ] Small Business Concern (37 CFR 1.9(f) and 1.27(c))
  [X] Non-Profit Organization (37 CFR 1.9(f) and 1.27(d))
  [ ] Non-Inventor Supporting Claim By Another
  [ ] Statement filed in prior application. Status still proper and desired
[ ] Preliminary Amendment
[ ] Change of Address

| CLAIMS AS FILED | | | | |
|---|---|---|---|---|
| For | Number Filed | Number Extra | Rate | Basic Fee $380.00 |
| Total Claims | 55-20 | 35 | x $ 9 = | 315.00 |
| Independent Claims | 7-3 | 4 | x $ 39 = | 156.00 |
| Multiple Dependent Claim Fee | | | x $130 = | 0.00 |
| TOTAL FILING FEE | | | | $851.00 |

[ ] Cancel in this divisional application original claims _____ of the prior application Serial No. _____ before calculating the filing fee. (At least one original independent claim must be retained for filing purposes.)

[X] A check in the amount of $891.00 to cover [X] filing fee and [X] assignment recordal fee ($40) is enclosed.
[X] Any deficiency or overpayment should be charged or credited to deposit account number 13-1703. A duplicate copy of this sheet is enclosed.

Customer No. 20575

Respectfully submitted,

MARGER JOHNSON & McCOLLOM, P.C.

Ariel Rogson
Reg. No. 43,054

MARGER JOHNSON & McCOLLOM, P.C.
1030 S.W. Morrison Street
Portland, Oregon 97205
(503) 222-3613

PATENT APPLICATION
Attorney Docket No. 6954-2

**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE**

Applicant or Patentee: Gregg Evan Rothermel, Margaret Myers Burnett and Lixin Li

Serial or Patent No.:

Filed or Issued:

For: METHODOLOGY FOR TESTING SPREADSHEETS

**DECLARATION CLAIMING SMALL ENTITY STATUS (37 CFR 1.9(f) and 1.27 (d)) NONPROFIT ORGANIZATION**

I hereby state that I am an official empowered to act on behalf of the nonprofit organization identified below.

NAME OF ORGANIZATION Oregon State University

ADDRESS OF ORGANIZATION A312 Kerr Administration Building
Corvallis, Oregon 97331-2140

TYPE OF ORGANIZATION

[X] University or other institution of higher education.
[] Tax exempt under IRS Code (26 USC 501(a) and 501(c)(3))
[] Nonprofit scientific or educational under statute of state of The United States of America (Name of state_____Citation of statute _____)
[] Would qualify as tax exempt under Internal Revenue Service Code (26 USC 501(a) and 501(c)(3)) if located in The United States of America
[] Would qualify as nonprofit scientific or educational under statute of state of The United States of America if located in The United States of America (Name of state _____
Citation of statute _____)

I hereby state that the nonprofit organization identified above qualifies as a non-profit organization as defined in 37 CFR 1 9(e) for purposes of paying reduced fees under Section 41(a) and (b) of Title 35, United States Code with regard to the invention entitled METHODOLOGY FOR TESTING SPREADSHEETS by inventor(s) Gregg Evan Rothermel, Margaret Myers Burnett and Lixin Li described in:
[X] the specification filed herewith.
[] application serial no. _____, filed _____
[] patent no._____, issued _____

I hereby state that rights under contract or law have been conveyed to and remain with the non-profit organization with regard to the above-identified invention.

If the rights held by the non-profit organization are not exclusive, each individual, concern or organization having rights to the invention is listed below* and no rights to the invention are held by any person, other than the inventor, who could not qualify as a small business concern under 37 CFR 1 9(d) or by any concern which would not qualify as a small business concern under 37 CFR 1.9(d) or a non-profit organization under 37 CFR 1.9(e) *NOTE: Separate statements are required from each named person, concern or organization having rights to the invention averring to their status as small entities. (37 CFR 1.27)

FULL NAME:
ADDRESS:
[ ] individual [ ] small business concern [X] nonprofit organization

I acknowledge the duty to file, in this application or patent, notification of any change in status resulting in loss of entitlement to small entity status prior to paying, or at the time of paying, the earliest of the issue fee or any maintenance fee due after the date on which status as a small entity is no longer appropriate (37 CFR 1.28(b))

NAME OF PERSON SIGNING: William Hostetler

TITLE OF PERSON OTHER THAN OWNER: Director of Technology Transfer

ADDRESS OF PERSON SIGNING: Oregon State University, A312 Kerr Administration Building, Corvallis, Oregon 97331-2140

Signature William Hostetler Date 10/25/99

PATENT APPLICATION
Attorney's Docket No. 6954-2

**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE**

Applicant or Patentee: Gregg Evan Rothermel, Margaret Myers Burnett and Lixin Li

Serial or Patent No.:

Filed or Issued:

For: METHODOLOGY FOR TESTING SPREADSHEETS

**DECLARATION CLAIMING SMALL ENTITY STATUS (37 CFR 1.9(f) and 1.27 (b)) INDEPENDENT INVENTOR**

As a below-named inventor, I hereby state that I qualify as an independent inventor as defined in 37 CFR 1.9(c) for purposes of paying reduced fees under Section 41(a) and (b) of Title 35, United States Code, to the Patent and Trademark Office with regard to the invention entitled METHODOLOGY FOR TESTING SPREADSHEETS described in the
[X] specification filed herewith
[ ] application serial no. _____, filed _____,
[ ] patent no. _____, issued _____

I have not assigned, granted, conveyed or licensed and am under no obligation under contract or law to assign, grant, convey or license, any rights in the invention to any person who could not be classified as an independent inventor under 37 CFR 1.9(c) if that person had made the invention, or to any concern which would not qualify as a small business concern under 37 CFR 1.9(d) or a nonprofit organization under 37 CFR 1 9(e).

Each person, concern or organization to which I have assigned, granted, conveyed, or licensed or am under an obligation under contract or law to assign, grant, convey, or license any rights in the invention is listed below:

[ ] no such person, concern, or organization
[X] persons, concerns or organizations listed below*

*NOTE: Separate statements are required from each named person, concern or organization having rights to the invention averring to their status as small entities. (37 CFR 1.27)

FULL NAME: Oregon State University
ADDRESS: A312 Kerr Administration Building, Corvallis, Oregon 97331-2140
[ ] individual [ ] small business concern [X] nonprofit organization

I acknowledge the duty to file, in this application or patent, notification of any change in status resulting in loss of entitlement to small entity status prior to paying, or at the time of paying, the earliest of the issue fee or any maintenance fee due after the date on which status as a small entity is no longer appropriate. (37 CFR 1.28(b))

**Full name of sole or first inventor:** Gregg Evan Rothermel

Inventor's signature: _____ 10/28/99 (Date)

**Full name of second co-inventor:** Margaret Myers Burnett

Inventor's signature: _____ 10/28/99 (Date)

**Full name of third co-inventor:** Lixin Li

Inventor's signature: _____ _____ (Date)

PATENT APPLICATION
Attorney's Docket No. 6954-2

**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE**

Applicant or Patentee: Gregg Evan Rothermel, Margaret Myers Burnett and Lixin Li

Serial or Patent No

Filed or Issued.

For: METHODOLOGY FOR TESTING SPREADSHEETS

**DECLARATION CLAIMING SMALL ENTITY STATUS (37 CFR 1.9(f) and 1.27 (b)) INDEPENDENT INVENTOR**

As a below-named inventor, I hereby state that I qualify as an independent inventor as defined in 37 CFR 1.9(c) for purposes of paying reduced fees under Section 41(a) and (b) of Title 35, United States Code, to the Patent and Trademark Office with regard to the invention entitled METHODOLOGY FOR TESTING SPREADSHEETS described in the
[X] specification filed herewith
[ ] application serial no _____, filed _____,
[ ] patent no. _____, issued _____

I have not assigned, granted, conveyed or licensed and am under no obligation under contract or law to assign, grant, convey or license, any rights in the invention to any person who could not be classified as an independent inventor under 37 CFR 1.9(c) if that person had made the invention, or to any concern which would not qualify as a small business concern under 37 CFR 1.9(d) or a nonprofit organization under 37 CFR 1.9(e).

Each person, concern or organization to which I have assigned, granted, conveyed, or licensed or am under an obligation under contract or law to assign, grant, convey, or license any rights in the invention is listed below.

[ ] no such person, concern, or organization
[X] persons, concerns or organizations listed below*

*NOTE: Separate statements are required from each named person, concern or organization having rights to the invention averring to their status as small entities. (37 CFR 1.27)

FULL NAME: Oregon State University
ADDRESS: A312 Kerr Administration Building, Corvallis, Oregon 97331-2140
[ ] individual [ ] small business concern [X] nonprofit organization

I acknowledge the duty to file, in this application or patent, notification of any change in status resulting in loss of entitlement to small entity status prior to paying, or at the time of paying, the earliest of the issue fee or any maintenance fee due after the date on which status as a small entity is no longer appropriate. (37 CFR 1.28(b))

**Full name of sole or first inventor:** Gregg Evan Rothermel

Inventor's signature: _____ _____ (Date)

**Full name of second co-inventor:** Margaret Myers Burnett

Inventor's signature: _____ _____ (Date)

**Full name of third co-inventor:** Lixin Li

Inventor's signature: _____ 11/02/99 (Date)

# METHODOLOGY FOR TESTING SPREADSHEETS

## RELATED APPLICATION DATA

This invention was made with the U.S. Government support under Grant Numbers CCR-9457473, CCR-9703108 and EIA-9806821, awarded by the National Science Foundation (NSF). The U.S. Government has certain rights in the invention.

This application claims priority from U.S. Provisional Application No. 60/125,414, filed March 22, 1999, entitled "METHODOLOGY FOR TESTING SPREADSHEETS" incorporated by reference.

## BACKGROUND OF THE INVENTION

Spreadsheet languages, which are also known as form-based languages in some of the research literature, provide a declarative approach to programming, characterized by a dependence-driven, direct-manipulation working model (Ambler92). Users of spreadsheet languages create cells, and define formulas for those cells. These formulas reference values contained in other cells and use them in calculations. When a cell's formula is defined, the underlying evaluation engine calculates the cell's value and those of other affected cells (at least those that are visible to the user), and displays new results.

Spreadsheet languages include, as a subclass, commercial spreadsheet systems. These systems are widely used by end users, for a variety of computational tasks. The spreadsheet paradigm is also a subject of ongoing research. For example, there is research into using spreadsheet languages for matrix manipulation problems (Viehstaedt92), for providing steerable simulation environments for scientists (Burnett94), for high-quality visualizations of complex data (Chi97), and for specifying full-featured GUIs (Myers91).

Despite the end-user appeal of spreadsheet languages and the perceived simplicity of the paradigm, research shows that spreadsheets often contain faults. For example, in an early spreadsheet study, 44% of "finished" spreadsheets still had errors (Brown87). A more recent survey of other such studies reported errors in 38% to 77% of spreadsheets at a similar stage

(Panko96). Of perhaps even greater concern, this survey also includes studies of "production" spreadsheets, those actually in use for day-to-day decision-making, and these results ranged from 10.7% to 90% having errors. A possible factor in this problem is the unwarranted confidence creators of spreadsheets seem to have in the reliability of those spreadsheets (Wilcox97).

In spite of this evidence, no discussion is found in the research literature of techniques for testing spreadsheets. In fact, there has been only a little work on testing in other paradigms that follow declarative models. In the domain of functional and dataflow programming, there has been work on specification-based testing (e.g., (Kuhn97, Ouabd95)), but creators of spreadsheets rarely employ formal specifications. There has also been some recent research (Azem93, Belli95, Luo92) that considers problems of testing and reliability determination for logic programs written in Prolog. The logic paradigm is like the spreadsheet paradigm in that both are declarative, but several features of the logic paradigm, such as the bidirectional nature of unification and backtracking after failure, are so different from the spreadsheet paradigm that the testing techniques developed for Prolog cannot be applied to the spreadsheet paradigm.

On the other hand, there has been extensive research on testing imperative programs (e.g., (Clarke89, Duesterwald92, Frankl93, Frankl88, Harrold88, Hutchins94, Laski83, Ntafos84, Offutt96, Perry90, Rapps85, Rothermel97a, Weyuker86, Weyuker93, Wong95)), and it is in this body of work that the methodology presented in this document has its roots. However, significant differences exist between the spreadsheet and imperative programming paradigms, and these differences have important ramifications for testing methodologies. These differences can be divided into three classes.

The first class pertains to evaluation models. Evaluation of spreadsheets is driven by data dependencies that exist between cells, and spreadsheets contain explicit control flow only within cell formulas. Thus, spreadsheets are more appropriately tested using adequacy criteria that are data-dependence-based than criteria that are strictly control-flow-based. The dependence-driven evaluation model also implies that evaluation engines have flexibility in the scheduling algorithms and optimization devices they might employ to perform computations. A methodology for testing spreadsheets must be compatible with such

mechanisms, and not rely upon particular evaluation orders or prevent optimizations based on value caching.

The second class of differences pertains to interactivity: spreadsheet systems are characterized by incremental visual feedback that is intertwined with the program construction process. The most widely-seen example of this is the "automatic recalculation" feature. This incremental visual feedback makes desirable use of testing methodologies that support an incremental input and validation process. For example, when a user changes a formula, the testing subsystem should provide feedback about how this affects the "testedness" of each visible portion of the program. This raises the issue of dealing with evolving spreadsheets while maintaining suitable response time.

The third class of differences pertains to users of spreadsheet languages. Imperative languages are most commonly used by professional programmers who are in the business of producing software. These programmers can be expected to know something about testing, and to place a high priority on doing a reasonably good job of testing. On the other hand, spreadsheet systems are used by a variety of users, many of whom are not professional programmers and have no interest in learning about formal testing methodologies. The goal is to provide at least some of the benefits of formal testing methodologies to these users.

Accordingly, a need remains for a suitable methodology for testing spreadsheets.

## SUMMARY OF THE INVENTION

To accommodate the evaluation models used with spreadsheets and the interactive process by which they are created, the methodology of the present invention is validation-driven and incremental. This is accomplished through the use of a test adequacy criterion that focuses on dependencies that influence validated output cells, and through the use of incremental program analysis. To accommodate the user base of these languages, an interface to the methodology is provided that does not require an understanding of testing theory. This is accomplished through a fine-grained integration with the spreadsheet environment to provide testing information visually.

One aspect of the invention is a method for testing a spreadsheet cell. Test elements are collected for the spreadsheet cell. In the preferred embodiment, the test elements are du-

associations. The cell's execution trace is tracked, and the subset of test elements that participated in the execution trace is marked as executed.

Another aspect of the invention is a method for providing a user with feedback of the testedness of the spreadsheet cells. The set of test elements for each spreadsheet cell is maintained, and the subset of the set of test elements that have been exercised by a test is identified. Using the sizes of the set of test elements and the subset of the set, a testedness measure is calculated for the cell, which is provided to the user.

## BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 shows an example spreadsheet on a computer display.

FIG. 2 shows a second example spreadsheet with formulae embedded in some of the spreadsheet cells.

FIG. 3 shows the example spreadsheet of FIG. 2 with the formulae and several cells hidden.

FIG. 4 shows a cell relation graph for the spreadsheet of FIG. 2.

FIG. 5 shows the example spreadsheet of FIG. 2 in a state of partial construction.

FIG. 6 shows a flowchart for the algorithm for collecting du-associations affected by a cell according to the invention.

FIG. 7 shows the partial construction of the example spreadsheet in FIG. 5 after the user validates the `minuteHand` cell according to the invention.

FIG. 7A shows the testedness measures of the du-associations of FIG. 7.

FIG. 8 shows a flowchart of the algorithm for "exercising" du-associations after the user validates a cell according to the invention.

FIG. 9 shows a flowchart of the algorithm for "unexercising" du-associations after a cell is invalidated according to the invention.

FIG. 10 shows a flowchart of the algorithm for batch-collecting du-associations for a group of cells according to the invention.

FIG. 11 shows the percentage of faults detected by du-association test suites for a variety of sample spreadsheets.

FIG. 12 shows a computer system for testing a spreadsheet according to the preferred embodiment of the invention.

FIG. 13 shows abstractions of the data structures used to store du-associations for a spreadsheet according to the invention.

FIG. 14 shows a flowchart of the algorithm for providing the user with visual feedback of the testedness measure for a cell according to the invention.

## DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

### I. BACKGROUND AND DEFINITIONS

#### A. Spreadsheet languages

Users of spreadsheet languages set up spreadsheets and specify their contents in order to program. The contents of a spreadsheet are a collection of cells; each cell's value is defined by that cell's formula; and as soon as the user enters a formula, it is evaluated and the result is displayed. The best-known examples of spreadsheet languages are found in commercial spreadsheet systems, but there are also many research systems (e.g., (Burnett98b, Chi97, Leopold97, Myers91, Smedley96, Viehstaedt92)) based on this paradigm.

Below are described examples of spreadsheets in the research language Forms/3 (Burnett98b). FIG. 1 shows a traditional-style spreadsheet used to calculate student grades in Forms/3. A typical spreadsheet 100 is composed of a number of rows 105 and a number of columns 110. Each cell 115 lies at the intersection of a row and a column. As shown with cell 120, a cell can span multiple columns; not shown, but equally possible, are cells that span multiple rows or span both rows and columns. Each cell 115 can contain different data: for example, cell 125 stores the name "Farnes, Joan" and cell 130 stores the name "Green, Matt." The spreadsheet lists several students, and several assignments performed by those students. The last row in the spreadsheet calculates average scores for each assignment, the rightmost column calculates weighted averages for each student, and the lower-right cell gives the overall course average (formulas not shown).

FIGs. 2 and 3 show how a user could construct a graphical clock in Forms/3. Consider clock 200's thirteen cells shown in FIG. 2 with their formulas, including two input cells (upper left) (cells 205 and 206) that could eventually be replaced with references to the system clock, one output cell (middle left) (cell 210 and formula 210a), and several cells used in intermediate calculations (right) (cells 215-224 and formulae 251a-224a). The term *input cell* refers to cells whose formulas contain only constants. The term output cell refers to cells

not referenced in any formula. After the programming is finished, the cells that calculate intermediate results can be hidden, and other cells rearranged, to reach the user view shown in FIG. 3.

This description considers a "pure" spreadsheet language model, which includes ordinary spreadsheet-like formulas such as those described by the grammar in Table 1, but need not include advanced programmer-oriented features such as macros, imperative sublanguages, and recursion. Table 1 reflects a subset of Forms/3, a language following this model. The subset shown uses ordinary spreadsheet formulas for both numeric and graphical computations; the figures presented in this specification were programmed using this subset. From this grammar, it is clear that the only dependencies between one cell and another are data dependencies. Because of this fact, cells can be scheduled for evaluation in any order that preserves these dependencies.

```
formula ::= BLANK | expr
expr ::= CONSTANT | CELLREF | ERROR | infixExpr |
         prefixExpr | ifExpr | composeExpr
infixExpr ::= subExpr infixOperator subExpr
prefixExpr ::= unaryPrefixOperator subExpr |
               binaryPrefixOperator subExpr subExpr
ifExpr ::= IF subExpr THEN subExpr ELSE subExpr |
           IF subExpr THEN subExpr
composeExpr ::= COMPOSE subExpr withclause
subExpr ::= CONSTANT | CELLREF | (expr)
infixOperator ::= + | - | * | / | AND | OR | = | ...
unaryPrefixOperator ::= NOT | ERROR? | CIRCLE | ...
binaryPrefixOperator ::= LINE | BOX | ...
withclause ::= WITH subExpr AT (subExpr subExpr) |
               WITH subExpr AT (subExpr subExpr)
```

**Table 1**

## B. Evaluation strategies for spreadsheet languages

The evaluation strategies used in spreadsheet languages, unlike those used in imperative languages, have a great deal of latitude regarding execution sequence, provided that all dependencies are preserved. Thus, evaluation in spreadsheets processes cells in the dataflow paths between input cells and output cells following the principles of either eager evaluation or lazy evaluation, with a variety of optimizations and variations possible. Eager



5,404,529

evaluation is driven by changes: whenever a value of cell *X* is changed, the change is propagated to every cell that is affected by the change. For example, if a user edits cell *X*'s formula, then if cell *Y* references cell *X* in its formula then cell *Y* is also recomputed, which in turn causes cells that refer to cell *Y* to be recomputed, and so on. Determining which cells are affected is usually done conservatively, i.e., from a static perspective.

In contrast to this strategy, lazy evaluation is driven by output: the first time a cell *X* is displayed, it is computed, and so is every cell that cell *X* needs. For example, if cell *X* is moved onto the screen through window manipulations, every cell that it needs is computed (and every cell that they need, and so on) in order to finally calculate cell *X*. Whether cell *X* "needs" cell *Y* is usually determined dynamically. For example, if cell *X*'s formula is "TRUE or *Y*", then the reference to cell *Y* will not be needed if the evaluation engine evaluates the first operand before the second. It has been shown that lazy evaluation produces the same answers as eager evaluation, provided that both terminate. However, lazy evaluation computes fewer cells.

Because spreadsheet languages are visual, keeping many cells on display automatically, usually at least some of the cell values are cached. This means that an evaluation engine also needs to keep track of which cached values are up-to-date if the user has started changing formulas. There are several methods for doing so (which are surveyed in (Burnett98a)), but their mechanism is not relevant to the issues in this document.

In spreadsheet languages, some cells will be on the screen and some will not. There are both static and dynamic mechanisms for determining which are on the screen. For example, in some languages it is possible to statically "hide" cells. In most spreadsheet languages, the user can also scroll or otherwise move cells on and off the screen at runtime through direct manipulation. Which cells are on-screen can influence the user's testing behavior, because they determine which input cells a user can notice and attend to, and which output cells the user can see. In the case of languages following lazy evaluation, which cells are on-screen also determines which cells will be computed, since lazy evaluation is output-driven.

### C. An abstract model for spreadsheets

Test adequacy criteria provide help in selecting test data and in deciding whether a program has been tested "enough." Test adequacy criteria have been well researched for imperative languages (e.g., (Clarke89, Frankl88, Ntafos84, Perry90)), where they are often defined on abstract models of programs rather than on code itself. An abstract model for spreadsheet languages is created, called a *cell relation graph* (CRG). A CRG is a pair (V, E), where V is a set of *formula graphs*, and E is a set of directed edges connecting pairs of elements in set V. FIG. 4 depicts the CRG for `Clock 200`. FIG. 4 shows the interrelation of each formula graph 205b, 206b, 210b, and 215b-224b: which formula graphs depend on which other formula graphs.

Each formula graph in set V models flow of control within a cell's formula, and is comparable to a control flow graph representing a procedure in an imperative program (Aho86, Rapps85). There is one formula graph for each cell in the spreadsheet. The process of translating an abstract syntax tree representation of an expression into its control flow graph representation is well known (Aho86); a similar translation applied to the abstract syntax tree for each formula in a spreadsheet yields that formula's formula graph. For example, FIG. 4 shows the formula graphs for each of the cells in `Clock 200`, delimited by dotted rectangles. In the formula graphs, nodes labeled "E" and "X" are *entry* and *exit* nodes, respectively, and represent initiation and termination of the evaluation of formulas. Nodes with multiple out-edges (represented as rectangles) are *predicate* nodes. Other nodes are *computation* nodes. Edges within formula graphs represent flow of control between expressions, and edge labels indicate the value to which conditional expressions must evaluate for particular paths to be taken. For example, formula graph 210b for `theClock` includes entry node 405 (numbered 55), exit node 410 (numbered 59), and nodes 415, 420, and 425 (numbered 56, 57, and 58, respectively). Formula graph 210b also includes edges 430, 435, 440, 445, and 450.

One form of expression supported by spreadsheets that can be seen in the grammar of Table 1 merits special attention. In spreadsheet languages, because there are only expressions and hence no statements, the `if` construct instantiates an expression instead of a statement, which allows "conditional subexpressions", i.e., `if` expressions included as subexpressions in other expressions. For example, a cell *C*'s formula can consist of the

expression `x + (if y=0 then 1 else 2)`. But what is the proper formula graph for such an expression? The approach of (Aho86) parses this expression into a pair of intermediate code statements (instead of expressions): (i) `if y=0 then tmp=1 else tmp=2` and (ii) `C = x + tmp`. A formula graph for this pair of statements consists of the nodes required to represent statement (i), followed by the node required to represent statement (ii), and these nodes include constructs not legal in formulas to represent assignment, variables, and statements. A disadvantage of this approach, especially to an audience of end users, is that if reasoning about the testedness of a cell or relationship is based in part upon constructs not allowed in spreadsheet formulas, feedback about the testedness of a particular cell or relationship might not be understandable to the user.

A second approach to modeling this expression is to use a "virtual cell" `tmp` to represent the computation of `if y=0 then 1 else 2` and treat *C*'s formula as consisting only of `x + tmp`. However, since this approach introduces a cell not actually present in the spreadsheet, feedback based on reasoning about `tmp` may still not be understandable to the user.

A third approach is to distribute operand `x` over the `if` expression, obtaining `if y=0 then x+1 else x+2` (in the formula graph, not in the user view), and model this like any other `if` expression. The third approach was selected for the present invention, as it is the most likely to be understandable by end users. Note also that, although conditional subexpressions are not expected to be used widely by spreadsheet users, they are supported by popular spreadsheet languages, and hence need to be supported by the spreadsheet testing methodology.

The set E of edges in the CRG, *cell dependence edges*, models dependencies between cells. FIG. 4 depicts these edges by dashed lines. Each edge encodes the fact that the destination cell refers to the input cell in its formula; thus, the arrows show direction of dataflow. Note that cell dependence information is typically available to evaluation engines within spreadsheet systems as a consequence of the need to evaluate formulas; thus, this information need not be specially calculated for use in CRGs.

A way to associate dynamic execution of formulas with CRG components is also required. Let *F* be a formula with formula graph $\overline{F}$, and let $F_e$ and $F_x$ be the entry and exit

nodes, respectively, of $\overline{F}$. An evaluation of F traverses a path through $\overline{F}$, beginning at $F_e$ and ending at $F_x$. This path is called the *execution trace* for that evaluation.

### D. Test adequacy criteria for spreadsheets

This abstract model is used to define several test adequacy criteria for use with spreadsheets.

First, criteria analogous to the statement and branch coverage criteria used in the testing of imperative programs are defined. Let *F* be a formula with formula graph $\overline{F}$. Test *t* exercises a node *n* in $\overline{F}$ if *F*'s execution trace, given *t*, includes *n*. A test suite *T* is *node-adequate* for spreadsheet *S* if, for each formula graph $\overline{F}$ in the CRG for *S*, for each dynamically executable node *n* in $\overline{F}$, there is at least one test in *T* that exercises *n*. Similarly, test *t* exercises an edge $(n_1, n_2)$ in $\overline{F}$ if *F*'s execution trace, given *t*, includes $(n_1, n_2)$. A test suite *T* is *edge-adequate* for spreadsheet *S* if, for each formula graph $\overline{F}$ in the CRG for *S*, for each dynamically executable edge $(n_1, n_2)$ in $\overline{F}$, there is at least one test in *T* that exercises $(n_1, n_2)$.

Node and edge adequacy criteria, however, do not explicitly consider interactions between cell formulas, and these interactions are a common input of errors in spreadsheets (Brown87). An alternative criterion, *cell-dependence adequacy*, focuses on such interactions. Let *A* and *B* be cells in spreadsheet *S*, with formula graphs $\overline{A}$ and $\overline{B}$, respectively, and let *B* be cell dependent on *A*, represented as edge $(\overline{A}, \overline{B})$ in the CRG for *S*. Test *t exercises* cell dependence edge $(\overline{A}, \overline{B})$ if *B*'s execution trace, given *t*, contains a node associated with an expression that references *A*. A test suite *T* is *cell-dependence-adequate* for spreadsheet *S* if, for each cell dependence edge $(x, y)$ in the CRG for *S*, there is at least one test in *T* that exercises $(x, y)$.

The cell dependence adequacy criterion requires coverage of some dependencies between cells. However, it does not require coverage of all such dependencies, and it does not explicitly consider the effects of the subexpression-level control dependencies that are created by predicate expressions. (Rothermel97b) explored analogues of the dataflow test adequacy criteria that have been defined for imperative programs (Frankl88, Laski83, Ntafos84, Rapps85). (Rothermel97b) showed that a criterion based on the *all-uses* dataflow

adequacy criterion (Frankl88, Rapps85), which relates test adequacy to interactions between occurrences of variables in the input code, can be particularly appropriate for spreadsheets because it exercises both interactions between cells and expressions within cell formulas.

The remainder of this description focuses on the use of this criterion, called the *du-adequacy* criterion, as the preferred test element. The criterion is described and defined, and then the testing methodology is illustrated based on the criterion. That methodology can easily be adapted to support the use of the other criteria described above as test elements.

### E. DU-adequacy for spreadsheets

In spreadsheets, cells serve as variables, and the value for cell $C$ can be defined only by expressions in $C$'s formula. Let $C$ be a cell in spreadsheet S, with formula $F$ and formula graph $\overline{F}$. If $C$ is an input cell, then $\overline{F}$ contains only one node other than entry and exit nodes, and that node is a *definition* of $C$. If $C$ is not an input cell, then each computation node in $\overline{F}$ that represents an expression referring to cell $D$ is a *c-use* (computation use) of $D$ and a *definition* of $C$. Each edge in $\overline{F}$ that has as its input a predicate node $n$ such that $n$ represents a conditional expression referring to another cell $D$ is a *p-use* (predicate use) of $D$.

A *definition-use association* (du-association) links definitions of cells with uses that those definitions can reach. Two types of associations are of interest. A *definition-c-use association* is a triple $(n_1, n_2, C)$, where $n_1$ is a definition of cell $C$, $n_2$ is a c-use of $C$, and there exists an assignment of values to $S$'s input cells, in which $n_1$ reaches $n_2$. A *definition-p-use association* is a triple $(n_1, (n_2, n_3), C)$, where $n_1$ is a definition of cell $C$, $(n_2, n_3)$ is a p-use of $C$, and there exists an assignment of values to $S$'s input cells, in which $n_1$ reaches $n_2$, and causes the predicate associated with $n_2$ to be evaluated such that $n_3$ is the next node reached.

A test adequacy criterion is *applicable* if, for every program $P$, there exists a finite test set that is adequate according to that criterion for $P$ (Weyuker86). To ensure the applicability of adequacy criteria based on these definitions, the definitions specify only *executable* du-associations: du-associations for which there exists some assignment of input values to S that causes the definition to reach the use. Determining whether a du-association is executable is, however, a difficult problem (Frankl88, Weyuker93); thus, algorithms for

calculating the du-associations that exist in a program approximate them by collecting *static du-associations*: du-associations that can be determined by static inspection of the code.

Given du-association $(n_1, n_2, C)$ (or $(n_1, (n_2, n_3), C)$), where $n_2$ (or $(n_2, n_3)$) is in $\overline{F}$, a test $t$ is said to *exercise* that du-association if the execution trace of $\overline{F}$, given $\underline{t}$, contains $n_2$ (or $(n_2, n_3)$), such that $n_1$ is the definition of cell $C$'s execution time given $\underline{t}$.

A notion of a du-adequate test set in terms of its inclusion of tests that exercise all du-associations could be defined. However, under such a definition, a test suite could be adequate even though no attempt had been made to validate test outputs. Thus, instead, following the notion of the "output-influencing all-du-pairs" criterion introduced in (Duesterwald92), a test adequacy criterion defined in terms of du-associations that affect cell outputs is preferred. A test suite $T$ is *du-adequate* for spreadsheet $S$ if and only if, for each du-association *du-a* in $S$, there is at least one test that exercises *du-a* in such a way that *du-a* influences, directly or transitively, a cell output.

Du-adequacy potentially integrates nicely into the spreadsheet paradigm, for several reasons. To satisfy the du-adequacy criterion, test suites must exercise interactions between cells that are not necessarily exercised by node-, edge-, or cell-dependence-adequate test suites. An additional advantage of this criterion involves its (relative) ease of application to spreadsheets. Dataflow analysis and dataflow testing of imperative programs are complicated by the presence of dynamically-determined addressing and aliases. Spreadsheets might utilize arrays and matrices and refer to them in formulas; however, for most spreadsheet languages, such references can be resolved statically. Spreadsheets can have aliases to the extent that multiple names refer to a single cell; however, for most spreadsheet languages these aliases, too, can be resolved statically. Finally, as shown below, the absence in spreadsheets of loops and of definitions that "kill" other definitions simplifies algorithms for calculating du-associations.

### F. A System for Testing

FIG. 12 shows a typical computer system 1200 that can be used for testing spreadsheet cells using the preferred embodiment. Computer system 1200 includes a computer 1205 (including, among other components, a CPU and memory (not shown)), a

monitor 1210, a keyboard 1215, and a pointer on mouse 1220. Optional equipment not shown can include a printer, plotter, light pen, and so forth. Computer system 1200 can also be connected to a network. Displayed on monitor 1210 is a spreadsheet 1225. Spreadsheet 1225 includes several cells 1230, 1235, and 1240. Cells 1235 and 1240 in particular show cells in various stages of testedness. In the preferred embodiment monitor 1210 displays cells 1235 and 1240 using colors; however, grayscale and fill patterns can also be used to indicate measures of testedness, and will be used in this application for simplicity.

## II. A METHODOLOGY FOR TESTING SPREADSHEETS

Three classes of differences between the spreadsheet language paradigm and traditional imperative paradigms were discussed earlier. To accommodate these differences, a testing methodology based on the use of the du-adequacy criterion that is validation-driven and incremental, and is integrated at a fine granularity into the spreadsheet environment, is developed, providing the following functionalities:

- The ability to determine incrementally the static du-associations in an evolving spreadsheet whenever a new cell formula is entered.
- The ability to track automatically the execution traces, which provide the information necessary to determine the dynamic du-associations that currently influence calculations.
- A user-accessible facility for pronouncing outputs "validated" at any point during spreadsheet development, and the abilities both to determine the du-associations that should be considered exercised as a result of this validation and to communicate immediately to the user how well exercised the visible section of the spreadsheet is.
- The ability to determine the du-associations affected by a spreadsheet change, and immediately depict their altered validation status in the visible section of the spreadsheet.
- The ability to recalculate du-associations and validation information when an entire pre-existing spreadsheet is loaded, or when a large portion of a spreadsheet is modified by a single user action.

How this methodology provides these functionalities to spreadsheet languages is next described in the context and sequence of a series of tasks in an integrated spreadsheet development and testing session.

**Task 1: Collecting static du-associations**

Suppose that, starting with an empty spreadsheet, the user begins to build the `Clock` application discussed earlier by entering cells and formulas, reaching the state shown in FIG. 5. In FIG. 5, spreadsheet 500 has cells 505, 510, 515, 520, and 525. Cells 515, 520, and 525 have formulae 515a, 520a, and 525a. Assume for the moment that the user does not change any formulas, but simply continues to add new ones. (This restriction is removed later.)

Because it would be expensive to exhaustively compute the du-associations for the entire spreadsheet after each new formula is added, these associations preferably are computed incrementally. Several algorithms for incremental computation of data dependencies exist for imperative programs (e.g., (Marlowe90, Pollock89)), and one of these algorithms could be adapted to this purpose. However, there are two attributes of spreadsheet systems that allow a more efficient approach.

First, in spreadsheet languages, the syntax of cell formulas ensures that all definitions of a cell *C* appear in *C*'s own formula, and none of these definitions may be "killed" by any other definition. Second, in spreadsheet systems, the evaluation engine must be called following each formula edit to keep the display up-to-date, visiting at least all cells that directly reference the new cell (termed the *direct consumers* of the new cell) and all visible cells that are directly referenced by (are the *direct producers* of) the new cell. At this time, the engine can record *local definition-use information* for the new cell, that is, the definitions and uses that are explicit in the cell's formula. Together, these two attributes mean that (static) du-associations can be incrementally collected following the addition of a cell *C* by associating all definitions in *C* with all uses of *C* in direct consumers of *C*, and associating all definitions in direct producers of *C* with all uses of those cells in *C*. (See [Marlowe90] for a different view of incremental computation of du-associations as applied within the imperative language paradigm.)

A hash table can efficiently store the following data for each cell *C* (abstractions of these data structures are shown in FIG. 13): *C.DirectConsumers* 1305, the cells that reference

*C*; *C.DirectProducers* 1310, the cells that *C* references; *C.LocalDefs* 1315, the local definitions in *C*'s formula; *C.LocalUses* 1320, the local uses in *C*'s formula; *C.ValidatedID* 1325a and *C.UnValidatedID* 1325b, integer flags whose use is described later; *C.DUA* 1330, a set of pairs (*du-association*, *exercised*) for each static *du-association* (d, u) such that u is in *C.LocalUses*, and *exercised* is a Boolean that indicates whether that association has been exercised; *C.Trace* 1335, which records dynamic trace information for *C*; and *C.ValTab* 1340, which records validation status. It is reasonable to rely upon the formula parser and the evaluation engine to provide the first four of these items, because they are already needed to efficiently update the display and cached value statuses after each edit. The remaining items can be calculated by the testing subsystem.

Algorithm `CollectAssoc` of Table 2 is triggered when a new formula is added, to collect new static du-associations. FIG. 6 shows a flowchart of algorithm `CollectAssoc`. Lines 2-5 (steps 605 and 610) collect du-associations involving uses in *C*. Lines 6-9 (steps 615 and 620) collect du-associations involving definitions (of *C*) in *C*. For example, referring back to FIG. 5, suppose that the most recent formula entered is that for cell `minuteY` 515. Note that its value is displayed, even though the spreadsheet has not been completely entered; when the evaluation engine is triggered to display this value, it collects *C.DirectConsumers*, *C.DirectProducers*, *C.LocalDefs*, and *C.LocalUses* for `minuteY` 515 (as done previously for the other cells on display when their formulas were entered). Called with cell `minuteY` 515, `CollectAssoc` employs this information to collect six new du-associations, described using the node numbers of FIG. 4 as: (2, (19, 20), `minute`), (2, (19, 21), `minute`), (2, 20, `minute`), (2, 21, `minute`), (20, 50, `minuteY`), and (21, 50, `minuteY`).

```
1. algorithm CollectAssoc(C)
2. for each cell D ∈ C.DirectProducers do
3.     for each definition d (of D) ∈ D.LocalDefs do
4.         for each use u of D ∈ C.LocalUses do
5.             C.DUA = C.DUA ∪ {((d, u), false)}
6. for each cell D ∈ C.DirectConsumers do
7.     for each use u of C ∈ D.LocalUses do
8.         for each definition d (of C) ∈ C.LocalDefs do
9.             D.DUA = D.DUA ∪ {((d, u), false)}
```

**Table 2**

`CollectAssoc` runs in time O(ud(DirectProducers + DirectConsumers)), where DirectProducers and DirectConsumers are the number of direct producers and direct consumers respectively of *C*, and u and d are the maximum number of uses and definitions, respectively, in those cells. In practice, u and d are typically small, bounded by the number of references in a single formula – usually less than 10. In this case, the algorithm's time complexity is O(DirectProducers + DirectConsumers).

This is no more than the order required by most evaluation engines' cell traversal needed to maintain a correct display and process cached values when a new formula is added – the event that triggers `CollectAssoc`. To see why this is true, consider the possible evaluation engine strategies. As discussed earlier, the two overall strategies possible are eager evaluation, in which the new formula's result is "pushed" to the cell's consumers, and lazy evaluation, in which visible cells "pull" results from their producers when needed for output. Hence, eager evaluation must read the direct producers in order to compute the result of the new formula, and must access the consumers (both direct and transitive) in order to push the new result along (which also ensures that the value cache is up-to-date). Lazy evaluation also must read the direct producers in order to compute the new result and display it, because a cell must be on the screen to be edited. However, there are several possibilities for how to manage the cache information, and this determines whether the direct consumers will be accessed. The most widely used value caching mechanism with lazy evaluation is *lazy evaluation with eager marking* (LazyEM), in which consumers of a newly-computed result are eagerly marked out-of-date (but not actually recomputed until needed, because evaluation itself is lazy). LazyEM thus, like eager evaluation, accesses all consumers (both direct and transitive). The LazyEM algorithm is formally presented and analyzed in (Hudson91), in which it is shown to be optimal in the number of computations performed, although not optimal in overhead.

Although these are the two evaluation approaches widely used for languages in this class, and both require at least O(DirectProducers + DirectConsumers) time, other approaches are possible that instead access consumers visible on the screen (both direct and transitive), which can be more or fewer than the total number of visible and non-visible direct consumers. See (Burnett98a) for a description and performance comparison of eager

evaluation and LazyEM, as well as several other variations on lazy evaluation, in the context of a class of visual languages that includes spreadsheet languages. The presence of these possibilities shows that O(DirectProducers + DirectConsumers) is not a lower bound for spreadsheet evaluation and cache maintenance. However, reports only of eager evaluation and LazyEM being actually used to respond to the trigger of a new formula entry have been found, and O(DirectProducers + DirectConsumers) is less than their time costs. This allows the cost of collecting the static du-associations to be potentially masked by the cost of evaluation activity, at least under these two widely-used evaluation approaches.

**Task 2: Tracking execution traces**

To track execution traces, which in Task 3 will enable the incremental computation of du-associations that have been exercised, it is sufficient to insert a probe into the evaluation engine. When cell *C* executes, this probe records the execution trace on *C*'s formula graph, storing it in *C.Trace*, adding only O(1) to the cost of execution. For example, in the case of `Clock`, at the moment depicted in FIG. 5, the execution trace stored for cell `minuteY` 515, described in terms of FIG. 4's node numbers, is (18, 19, 20, 22). If the cell is subsequently reevaluated, the old execution trace is replaced with the new one. Storing only the most recent execution trace in *C.Trace* is sufficient for coverage computation because the cumulative coverage in *C.DUA* is updated incrementally during validation, as described in the discussion of Task 3.

This approach functions for all varieties of evaluation engines: whether the engine eagerly or lazily evaluates cells, following any new formula entry and any dependence-preserving evaluation sequence, all cells have associated with them their most recent execution trace. The correctness of this approach for any evaluation engine rests upon two attributes of this strategy. First, each cell's execution trace is a set rather than a list, rendering any difference in order of execution irrelevant. Second, the granularity of the nodes in the formula graph (and hence in the execution trace) is coarser than the level at which subtle differences among evaluation approaches occur.

To illustrate the second attribute, consider execution of *C*. *C*'s entry point and exit point (both of which are unique) are by definition always executed. If *C*'s interior (non-entry/exit) nodes do not contain any conditionals ("if" expressions) then there is only one

interior node, which contains the entire expression and which, because it is executed at least in part, is recorded in the trace. On the other hand, if there are conditionals, the strategy is the same, eager or lazy – execute the condition node and either the "then" or the "else" node but not both. (Although it may seem that eager evaluation would execute both the "then" and the "else", this could lead to run-time errors, and hence even eager approaches employ "short circuit" evaluation to execute conditionals. An example of a formula that must be executed lazily to avoid such run-time errors is "`if A1=0 then 0 else 10/A1`".)

Note that, due to the fact that the execution traces are stored for each cell, an evaluation engine's particular caching optimizations do not change the execution traces. If cell *D* is executed because cell *C*'s execution requires it, there are only two possibilities: either *D* has a cached value or it does not. If *D* has a cached value, then *D*'s stored execution trace is still up-to-date. If *D* does not have a cached value, then *D* will be executed and its execution trace stored. In neither case does the execution of *D* affect the execution trace of *C*. Hence, whether *D* was executed now or at some previous time does not change the trace stored for *C* or the trace stored for *D*.

**Task 3: Pronouncing outputs "validated"**

How the data collected in Tasks 1 and 2 can be used to provide test adequacy information to the user in a way requires no understanding of formal notions of testing. Visual devices are used to draw attention to untested sections of the evolving spreadsheet.

In the desktop clock programming scenario, suppose that the user looks at the values displayed on the screen and decides that the `minuteHand` cell contains the correct value. To document this fact, the user clicks on the validation tab in the upper right corner of that cell. As FIG. 7 shows, one immediately-visible result of this action is the appearance of a checkmark in the validation tab 705. If the user enters another input in cell `minute` 510, `minuteHand`'s validation checkmark changes to a question mark (not shown in the figure), which means the current value has not been validated but some previously-displayed value has been. (Any evaluation engine must visit at least on-screen consumers of the new input to keep the displayed values up-to-date, so changing the checkmark to a question mark during these visits adds only O(1) to the cost of each.) The third possible appearance, a blank validation tab, means no validations have been done since the last formula change to *C* or to a

non-input cell affecting *C*. Thus, the validation tab keeps the user apprised of which cells have been explicitly validated and which have not, given the current collection of formulas.

A finer-grained device for communicating testing status involves test adequacy. Whenever a du-association participates in the production of a validated value, the *exercised* flag for that du-association (the second item of data kept for each du-association in the *.DUA* set for the cell in whose formula the use occurs) is set to "`true`". A "testedness" measure, such as the percentage, is then calculated of the du-associations whose uses occur in the cell, that have been exercised. This percentage is used to determine the cell's border color on a continuum from red (untested) to blue (100% of the du-associations whose uses occur in the cell having been exercised). (In a black and white display system, the continuum is shown as light gray to black.) With each validation that exercises a previously unexercised du-association, the border becomes less red (darker in these figures), indicating a greater degree of "testedness" for that cell. This visual feedback appears in all cells that contributed to the computation of the value in the validated cell.

In the example shown in FIG. 7, after the user has validated cell `minuteHand` 525 by checking the validation tab 705, the border 710 of cell `minuteHand` 525 turns more blue, because some of the nodes and edges in the formula graph for cell `minuteHand` 525 have been adequately tested. The computation of `minuteHand`'s 525 value involves two of the four du-associations that end in `minuteY` 515, two of the seven du-associations that end in `minuteHand` 525, and four of the 13 du-associations that end in `minutex` 520. Thus, after the user validates `minuteHand` 525, the cell borders are darkened using these fractions. Input cells are, by definition, fully exercised.

When borders are entirely blue, the user can see that each cell reference pattern (du-association) has been tested (i.e., executed with validation) at least once. As FIG. 7A shows, the user can also display arrows 715, 720, 725, 730, 735, and 740 that indicate cell reference patterns (du-associations). When formulas are showing, arrows point from definition to use (arrow 740). When a formula is not showing, the arrows pointing from or to definitions or uses, respectively, in that cell, join together (arrow 715). Arrows can also point from a definition to a cell, or from a cell to a use (all other arrows).

The cell reference pattern arrows 715, 720, 725, 730, 735, and 740 are colored using the same scheme as the borders, and thus, make explicit precisely which cell reference

patterns still need to be tested. In the current embodiment, when formulas are showing, arrows pointing from definition to c-use will be in one of two states: untested or tested; arrows pointing from definition to p-use will be in one of three states: untested, partially tested, or fully tested. (The partially tested state occurs where only one branch of the `if` construct has been executed.) For example, in FIG. 7A, cell reference pattern arrows 730 and 735 are untested, cell reference pattern arrow 720 is partially tested, and cell reference pattern arrows 725 and 740 are fully tested. However, a person skilled in the art will recognize that, depending on the spreadsheet constructs, cell reference pattern arrow coloration can vary on a continuum from untested to tested. For example, a spreadsheet that allowed constructs similar to the `switch` construct of the C programming language could have more than just three testedness states. When a formula is not showing, arrows are colored like the borders of the cells to which they point.

Table 3 displays the algorithm `Validate`, which is invoked when the user displayed value. FIG. 8 shows a flowchart of algorithm `Validate`. The algorithm uses the static du-association information and execution traces, previously calculated and stored as discussed in the descriptions of Tasks 1 and 2, to calculate the du-associations that participate in the production of *C*'s current value, and to update borders of participating cells. As the algorithm proceeds, it adds to stored *.DUA* data that indicates the du-associations that have been validated thus far. This coverage information is accumulated and retained across a succession of tests, even though cell execution traces change as subsequent tests are applied.

```
1.  algorithm Validate(C)
2.  ValidatedID = ValidatedID + 1
3.  C.ValTab = "checkmark"
4.  ValidateCoverage(C)

5.  procedure ValidateCoverage(C)
6.  C.ValidatedID = ValidatedID
7.  for each use u ∈ C.Trace do
8.      D = the cell referenced in u
9.      d = the current definition of D found in D.Trace
10.     C.DUA = C.DUA ∪ {((d, u), true)} - {((d, u), false)}
11.     if D.ValidatedID < ValidatedID then
12.         ValidateCoverage(D)
13. UpdateDisplay(C)
```

**Table 3**

Variable *ValidatedID*, referenced in the algorithm, can be set to 0 when the spreadsheet environment is first activated. Then, when cells are created or added to the spreadsheet, their *.ValidatedID* fields are initialized to 0. On each invocation of `Validate`, *ValidatedID* is incremented (line 1). The *.ValidatedID* fields for all cells visited are assigned this value of *ValidatedID*, which prevents duplicate visits to the same cell. By using an integer rather than a Boolean, and incrementing it on each invocation of the algorithm, the need to initialize the flag for all cells in the spreadsheet on each invocation is avoided. It is assumed that *ValidatedID* will not overflow, to simplify the presentation.

In lines 2-4 (step 805), the cell the user has identified as validated is marked accordingly and *ValidatedID* is updated. At line 6 (step 808), the validated cell's validation is updated. At line 7 (step 810), each use in the trace for the validated cell is considered. In lines 8-9 (step 815), the referenced cell is identified, as is the definition in the current trace for the referenced cell. Line 10 (step 820) marks the du-association exercised. Lines 11-12 (step 825) test to see if the referenced cell has been updated since the current cell was validated and, if not, recursively exercises the du-associations for the referenced cell. Finally, line 13 (step 830) updates the display for the current cell by changing the border color according to the percentage of du-associations for the current cell that have been exercised, and to reflect the validated status of the current cell.

The use of *ValidatedID* ensures that `ValidateCoverage` is called no more than once per cell, and that `Validate` terminates in a worst-case time proportional to the number

of du-associations validated. Because the set of uses in a cell's trace corresponds to a set of definitions in that cell's direct producers, which in turn lead to that cell's indirect producers, the cost of validation is bounded by the number of direct and transitive producers of a cell. This is less than or equal to the cost of calculating the cell's value the first time (when no reusable values are present in the cache), but the algorithm is triggered by a user interaction that does not require evaluation, so, unlike the other algorithms presented, its cost cannot be masked by the cost of the evaluation process.

A generalization of this algorithm related to the approach of (Duesterwald92) uses slicing to locate the expressions that contribute to the computation of the validated output, and identifies the du-associations involved in the computation from that slice. This generalized approach is applicable to languages with recursion, iteration, and redefinitions of variables. In the absence of such features, the more efficient approach that presented here suffices.

**Task 4: Adjusting test adequacy information**

So far, focus has been limited to how the methodology of the invention handles cell formulas as they are added to a spreadsheet. Another aspect of the methodology addresses the other basic edits possible with spreadsheets, namely, deleting a cell or changing a cell's formula. Changes to a constant-formula cell are equivalent to the application of a new test input (which may or may not be followed by validations by the user), and requires no action beyond that involved in recalculating execution traces as discussed under Task 2. Deletion of a cell is equivalent to modifying that cell's formula to BLANK. Thus, only modifications to non-constant formulas need be considered.

Suppose that the user has done quite a bit of testing, and has discovered a fault that requires a formula modification with far-reaching consequences. The user may believe that the spreadsheet is still fairly well tested, and not realize the extent to which the modification invalidates previous testing.

To address this lack of awareness, a system must immediately reflect the new test adequacy status of the spreadsheet whenever a cell is modified. In this context, the problem of interactive, incremental testing of spreadsheets resembles the problem of regression testing imperative programs, and one could adapt techniques for incremental dataflow analysis (e.g.,

(Marlowe90, Pollock89)) and incremental dataflow testing of imperative programs (e.g., (Gupta93, Harrold88, Rothermel94)). Such an approach would apply to spreadsheet languages in which cell references can be recursive or in which formulas contain iteration. In the absence of such features however, the more efficient approach presented here suffices.

To reflect the new test adequacy status of the spreadsheet whenever a cell is modified, the system must (1) update *C*'s static du-association and dynamic execution trace information, and (2) update the *exercised* flags on all du-associations affected by the modification, allowing calculation and display of new border and arrow colors to reflect the new "testedness" status of the spreadsheet. Validation tab statuses on cells that were dependent on (consumers of) *C* must also be adjusted, changing all checkmarks to question marks if the cell retains any exercised du-associations after affected associations have been reset, or to blank if all the cell's exercised flags are now unset. (Because all of *C*'s du-associations have been reset, the validation tab for *C* is changed to blank.) For example, in the completed `Clock` spreadsheet of FIG. 2, if the user changes cell `minutex`'s 215 formula, then the du-associations involving `minutex` 215, and the validation statuses for `minutex` 215, `minuteHand` 222, and `theClock` 210 must all be adjusted to blank. On the other hand, if the user changes the value of `minute` 205, the validation statuses for cells `minutex` 215, `minutey` 216, `hourx` 217, `houry` 218, `hourWithFraction` 219, `fraction` 220, `hourHand` 221, `minuteHand` 222, and `theClock` 210 are adjusted to question marks or to blanks, depending on the cells' other du-associations' exercised statuses.

The preferred embodiment handles item (2) first, removing the old information before adding the new. Let *C* be the modified cell. A conservative approach that recursively visits potentially affected cells is used. The algorithm, `UnValidate`, given in Table 4, is similar to `Validate`, but instead of using dynamic information to walk backward through producers, it uses static information to walk forward through consumers. FIG. 9 shows a flowchart of algorithm `UnValidate`. As the algorithm walks forward, it changes the *exercised* flag on each previously exercised du-association it encounters to "`false`", and keeps track of each cell visited in *AffCells*. On finishing the work for all the cells, the algorithm updates the border and arrow colors and validation tab for each cell in *AffCells*.

```
1.  algorithm UnValidate(C)
2.  global AffCells = {}
3.  UnValidatedID = UnValidatedID + 1
4.  UnValidateCell(C)
5.  for each cell D ∈ AffCells do
6.      UpdateValTab(D)
7.      UpdateDisplay(D)

8.  procedure UnValidateCell(E)
9.  E.UnValidatedID = UnValidatedID
10. AffCells = AffCells ∪ E
11. for each cell F ∈ E.DirectConsumers do
12.     for each definition d (of E) in E do
13.         for each ((d, u), true) ∈ F.DUA do
14.             F.DUA = F.DUA ∪ {((d, u), false)} - {((d, u), true)}
15.     if F.UnvalidatedID < UnValidatedID then
16.     UnValidateCell(F)
```

**Table 4**

Variable *UnValidatedID*, referenced in the algorithm, can be set to 0 when the spreadsheet environment is first activated. Then, when cells are created or added to the spreadsheet, their *.UnValidatedID* fields are initialized to 0. On each invocation of `UnValidate`, *UnValidatedID* is incremented (line 3). The *.UnValidatedID* fields for all cells visited are assigned this value of *UnValidatedID*, which prevents duplicate visits to the same cell. By using an integer rather than a Boolean, and incrementing it on each invocation of the algorithm, the need to initialize the flag for all cells in the spreadsheet on each invocation is avoided. It is assumed that *UnValidatedID* will not overflow, to simplify the presentation.

In line 2 (step 905), a data structure for storing the affected cells is initialized. This data structure may be a set, a list, or any other convenient storage mechanism. In line 3, the variable *UnValidatedID* is incremented. In line 4, the current cell has its du-associations marked as unexercised. This process comprises steps 910-930. In line 9 (step 910), the current cell is marked as no longer valid. In line 10 (step 915), the current cell is added to the data structure storing the affected cells. In lines 11-14, for each cell that depends on the current cell (step 920), the consuming cell's du-association set is updated so that each du-association for the consuming cell that uses a definition of the current cell is marked as

unexercised (step 925). Lines 15-16 (step 930) test to see if the consuming cell has been updated since the current cell was invalidated and, if not, recursively marks as unexercised the du-associations for the consuming cell. Finally, lines 5-7 update the validation tabs for the affected cells (step 935) and update the display for the affected cells. Updating the display includes changing the border and arrow colors according to the percentage of du-associations for the affected cells, and to reflect the validated status of the affected cells.

At this point, the static du-association and dynamic trace information stored with *C* can be updated. First, all stored static du-associations involving *C* are deleted; these du-associations are found in the information stored for *C* and for cells in *C.DirectConsumers*. This deletion also guarantees that du-associations that end in *C* are no longer marked "exercised." Having removed the old du-associations, it is necessary only to re-invoke `CollectAssoc` as described earlier to add new associations. Finally, stored execution traces are automatically updated via the evaluation engine as described earlier.

Because `UnValidate`'s processing is consumer-driven, then as with Task 1, the cell visits required by it are already required for display and value cache maintenance under eager evaluation and under LazyEM, but not necessarily by other evaluation engines that might be possible. However, in the cases of eager evaluation and LazyEM, the time cost of the algorithm increases only by a constant factor the cost of other work being performed by the environment when a formula is edited.

**Task 5: Batch computation of information**

Test information can be saved when a spreadsheet is saved. Then, when the spreadsheet is reloaded later for further development, it is not necessary to analyze it exhaustively to continue the testing process. Still, there are some circumstances in which it might be necessary to calculate static du-association information for a whole spreadsheet or section of a spreadsheet: for example, if the user does a block copy/paste of cells, or imports a spreadsheet from another environment that does not accumulate the necessary data. One possible response to such an action is to iteratively call the `CollectAssoc` algorithm presented earlier for each cell in the new spreadsheet section. This approach, however, may visit cells more times than necessary.

Table 5 presents a more efficient approach, `BatchCollectAssoc`, that takes an entire set U of cells as input, collects (from the spreadsheet environment) the set V of cells that are direct producers of cells in U, and then makes a single pass over V and U to gather du-associations. FIG. 10 shows a flowchart of algorithm `BatchCollectAssoc`. Although this algorithm has the same worst-case runtime as `CollectAssoc`, when there are interrelationships among the cells, its set-driven approach allows it to eliminate some duplicated visits to cells.

```
1.  algorithm BatchCollectAssoc(U)
2.  V = {}
3.  for each cell C ∈ U do
4.      V = V ∪ C.DirectProducers
5.  for each cell C ∈ V ∪ U do
6.      for each cell D ∈ C.DirectConsumers do
7.          for each definition d (of C) ∈ C.LocalDefs do
8.              for each use u (of C) ∈ D.LocalUses do
9.                  D.DUA = D.DUA ∪ {((d, u), false)}
```

**Table 5**

In lines 2-4 (step 1005), every cell that is a direct producer of a cell in the input set ($u$) is identified. These newly identified cells augment the input set for the remainder of the algorithm (step 1010). In lines 5-9, each consumer cell of any definition in the augmented set is identified (step 1015), and an unexercised du-association is added to the using consumer cell (step 1020).

Another approach to this problem would be to propagate definitions forward, and uses backwards, across flow and cell dependence edges in the CRG. This approach applies to spreadsheet languages in which cell references can be recursive or in which formulas contain iteration or redefinitions of variables. In the absence of such features, the more efficient approach presented here suffices.

**Visual Representation Devices**

The visual representation of testedness used in this document reflects three constraints on the visual representation that are believe to be important for integration into spreadsheet

environments. These constraints are in part drawn from literature on cognitive aspects of programming (Green96, Yang97). The constraints upon the visual representation are that first, the visual representation should be frugal enough of screen space that it does not significantly decrease the number of cells that can be displayed. Second, the visual representation should maintain consistency with the current formulas in the visible cells. Third, the visual representation devices should be accessible to as wide an audience as possible, including those with mild visual impairments.

The prototype used to create the figures in this document used the following representation devices to satisfy these constraints. To satisfy the first constraint, testedness information is enclosed with cell border colors, rather than displaying it in textual summaries, and explicit validation status is enclosed with the presence or absence of checkmarks or question marks. However, a person skilled in the art will recognize that other mechanisms can be used to convey this information.

To satisfy the second constraint, the display of testedness is automatically updated whenever any action by the user or by the system changes testedness status. Thus, outdated testedness information is never left on the screen.

To satisfy the third constraint, border colors along a red-blue continuum are selected to be used against the usual white background for spreadsheets. The colors red and blue are easily discriminated (Christ75) and because, due to the physiology of the human eye, red stands out while blue recedes (Shneiderman98), they draw users' attention to untested cells. The blue can also be desaturated to enhance this effect. Also, because red and blue differ in two of the three RGB components of screen color, this device should be usable by some red-deficient or blue-deficient users (Murch84); the gray-black continuum shown in the drawings is also an option for color-deficient users.

An additional visual device that can be employed to emphasize the differences among the three categories of no coverage validated, partial coverage validated, and all coverage validated, is to use a quadratic formula that separates the 0% and 100% colors from the partial coverage colors. Of course, while this furthers the goal of drawing overall attention to untested cells, exact degree of coverage for a particular cell is not likely to be readily discernable by most users. An additional optional thermometer-like indicator can be used along a border for when a user wishes to see more exact information about a cell's coverage.

In FIG. 12, the color red is represented by a fill pattern of diagonal lines running from the top right to the bottom left. The color blue is represented by a fill pattern of diagonal lines running from the top left to the bottom right. Thus, cell 1235 is as yet completely untested, cell 1240 is partially tested, and cell 1245 is completely tested.

With reference to FIG. 14, the testedness measure is visually represented as follows: At step 1405, the total number of du-associations for the cell is determined. Then at step 1410, the percentage of exercised du-associations for the cell is calculated. Finally, at step 1415, the border color for the cell is changed to reflect the testedness measure.

## III. EMPIRICAL RESULTS

To empirically test whether du-adequate testing will reveal a reasonable percentage of faults in spreadsheets, a prototype of the preferred embodiment within the Forms/3 environment has been implemented. The prototype incorporates the algorithms described herein. The screen shots used in the drawings are from this prototype. The prototype was exercised to perform an empirical study of the effectiveness of du-adequate test suites at detecting faults.

### A. Methodology

This study employed eight Forms/3 spreadsheets (see Table 6) from experienced Forms/3 users. Three of the spreadsheets (TimeCard, Grades, and Sales) are modeled after spreadsheets written in commercial spreadsheet systems, and the others are spreadsheets written in a research spreadsheet language: two are simple simulations (FitMachine and MicroGen), one a graphical desktop clock (Clock), one a number-to-digits splitter (Digits), and the last a quadratic equation solver (Solution).

| Spreadsheet | Expression | Du-associations | Version | Test Pool Size | Average Test Suite Size |
|---|---|---|---|---|---|
| Clock | 33 | 64 | 7 | 250 | 11.3 |
| Digits | 35 | 89 | 10 | 230 | 22.7 |
| FitMachine | 33 | 121 | 11 | 367 | 30.2 |
| Grades | 61 | 55 | 10 | 80 | 9.8 |
| MicroGen | 16 | 31 | 10 | 170 | 10.4 |
| Sales | 30 | 28 | 9 | 176 | 10.4 |
| Solution | 20 | 32 | 11 | 99 | 12.0 |
| TimeCard | 33 | 92 | 8 | 240 | 16.7 |

**Table 6**

Seven users experienced with Forms/3 and commercial spreadsheets were asked to insert faults into the subject spreadsheets which faults, in their experience, are representative of faults found in spreadsheets. A Forms/3 user who had no knowledge of these specific faults was then asked to generate a pool of tests for each of the base versions of the subject spreadsheets. For each base spreadsheet, this user first created tests of spreadsheet functionality. He then executed these tests on the base spreadsheet to determine whether together they exercised all executable du-associations in the spreadsheet, and generated additional tests to ensure that each executable du-association in the spreadsheet was exercised by at least five tests in the test pool. He also verified that for all tests, validated cells in the base version produced correct values.

These test pools were used to create du-adequate test suites for the subject spreadsheets. First, for each test $t$ in the test pool, the du-associations exercised by $t$ were determined. Then test suites were created by randomly selecting a test, adding it to the test suite only if it added to the cumulative coverage achieved by tests in that suite thus far, and repeating this step until coverage was du-adequate. Between 10 and 15 du-adequate test suites were generated for each of the subject spreadsheets; Table 6 lists the average sizes of these test suites.

Each base version $S$ was known to produce correct output, and only a single fault was inserted in each faulty version. This made it possible to determine whether a fault had been revealed in a modified version $S'$ by a test suite $T$ simply by comparing the validated output of $S$ (the output which, for that test, was confirmed by the tester to be correct) for each test $t$ in $T$ with the validated output of $S'$ on $t$. Thus, to obtain fault detection results, for each base

version $S$, with its faulty versions $S_1 \ldots S_k$ and universe $U$ of test suites, for each test suite $T$ in $U$:

1. all tests in $T$ on $S$ were run, saving outputs,
2. for each modified version $S_i$ of $S$:
   (a) all tests in $T$ on $S_i$ were run, saving outputs,
   (b) $T$ was recorded as fault-revealing for $S_i$ if and only if the output of the validated cell for some test $t$ in $T$ executed on $S_i$ differed from the output of that cell when t was executed on $S$.

### B. Data and analysis

FIG. 11 displays fault detection data, using box plots to show, for each spreadsheet, the percentage of faults detected by the du-adequate test suites. Dashed crossbars represent median percentages of faults detected over the set of test suites for the spreadsheet. The boxes show the ranges of percentages in which half of the fault detection results occurred. The whiskers that extend below and above boxes indicate ranges over which the lower 25% and upper 25% of the data, respectively, occurred.

The overall average (mean) percentage of faults detected for all spreadsheets, faulty versions, and test suites in the study was 81%. Fault detection varied across spreadsheets, but in all but one case (on two versions of TimeCard) exceeded 50%. Although differences in experimental instrumentation make comparisons difficult, this fault-detection effectiveness is comparable to or better than the effectiveness demonstrated by the all-uses criterion in studies of imperative programs (Frankl93, Hutchins94, Offutt96, Weyuker93, Wong95).

One additional cost factor of dataflow testing to consider involves static du-associations that are recognized by the testing system's analysis, but that are nonexecutable. On average, 11.65% of the static du-associations calculated by the algorithms for the subject spreadsheets were nonexecutable. This rate is lower than the average rates of 26% and 27% observed in two studies of imperative programs reported in (Weyuker93). Nevertheless, the presence of these du-associations could be difficult to explain to users. Future work will consider whether techniques for determining (approximately) path feasibility (e.g., (Clarke76)) can operate cost-effectively behind the scenes to address this problem.

In related empirical work performed involving human subjects (detailed in (Cook99)), several measurements of users' testing effectiveness and efficiency were significantly higher for subjects using Forms/3 supplemented by this testing methodology than for those using Forms/3 without the testing methodology.

## IV. CONCLUSION

Due to the popularity of commercial spreadsheets, spreadsheet languages are being used to produce software that influences important decisions. Furthermore, due to recent advances from the research community that expand its capabilities, the use of this paradigm is likely to continue to grow. The fact that such a widely-used and growing class of software often has faults should not be taken lightly.

To address this issue, the methodology of the present invention brings some of the benefits of formal testing to spreadsheet software. Important to its appropriateness for the spreadsheet paradigm are four features. First, the methodology accommodates the dependence-driven evaluation model, and is compatible with evaluation engine optimizations, such as varying evaluation orders and value caching schemes. Second, the set of algorithms used is structured such that their work can be performed incrementally, and hence can be tightly integrated with the highly interactive environments that characterize spreadsheet programming. Third, the algorithms are reasonably efficient given their context, because the triggers that require immediate response from most of the algorithms also require immediate response to handle display and/or value cache maintenance, and the same data structures must be traversed in both cases. The only algorithm that adds more than a constant factor is `Validate`, whose cost is the same order as the cost of recalculating the cell being validated. Finally, the methodology does not require user knowledge of testing theory; instead, the algorithms track the "testedness" of the spreadsheet incrementally, and use visual devices to call attention to insufficiently tested interactions.

Empirical results suggest that the methodology can achieve fault detection results comparable to those achieved by analogous techniques for testing imperative programs. Further experiments involving human subjects, showed that users were both significantly more effective and significantly more efficient at testing when supported by the methodology

than when not supported by it. These results are important, because they imply that the potential benefit of this approach to spreadsheet users may be substantial.

Having illustrated and described the principles of our invention in a preferred embodiment thereof, it should be readily apparent to those skilled in the art that the invention can be modified in arrangement and detail without departing from such principles. We claim all modifications coming within the spirit and scope of the accompanying claims.

6954-2.app.doc

We claim:

1. In a computerized spreadsheet which includes multiple spreadsheet cells, a method for testing a first spreadsheet cell containing a first formula, the method comprising:

creating a first formula graph for the first formula of the first spreadsheet cell, the first formula graph containing nodes representing predicates and definitions in the first formula of the first spreadsheet cell and edges representing execution flow between the nodes of the first formula graph;

collecting one or more du-associations for the first spreadsheet cell, where each du-association relates a node in a second formula graph for a second formula of a second spreadsheet cell with one of a node or edge in the first formula graph of the first spreadsheet cell;

tracking an execution trace for the first spreadsheet cell, where the execution trace comprises a sequence of nodes in the first formula graph of the first spreadsheet cell and a set of edges from the first formula graph connecting pairs of the nodes in the execution trace sequence; and

after receiving a user validation of the first spreadsheet cell, marking as exercised the du-associations containing a node or edge in the execution trace for the first spreadsheet cell.

2. A method according to claim 1 further comprising repeating the steps of creating, collecting, tracking, and marking for each second spreadsheet cell with an exercised du-association so that any du-associations for each second spreadsheet cell are marked as exercised after at least one of the first spreadsheet cell and second spreadsheet cell has been user-validated.

3. In a computerized spreadsheet containing a plurality of spreadsheet cells, a method for testing a first spreadsheet cell containing a formula, the method comprising:

collecting one or more test elements for the first spreadsheet cell;

tracking an execution trace of the first spreadsheet cell; and

marking as exercised a subset of the test elements that participated in the execution trace for the first spreadsheet cell.

4. A method according to claim 3 further comprising:

identifying all other spreadsheet cells upon which the first spreadsheet cell depends; and

for each other spreadsheet cell, repeating the steps of collecting, tracking, and marking.

5. A method according to claim 3 wherein collecting one or more test elements includes collecting one or more du-associations for the first spreadsheet cell.

6. A method according to claim 5 wherein one or more du-associations are collected as the formula is entered into the first spreadsheet cell.

7. A method according to claim 5 wherein collecting one or more du-associations includes:

identifying all other cells used in the formula of the first spreadsheet cell;

identifying all definitions of the other cells that are used in the formula of the first spreadsheet cell; and

creating a du-association between each definition of the other cells and each use of the definition in the formula of the first spreadsheet cell.

8. A method according to claim 3 wherein tracking an execution trace includes tracking which parts of the formula for the first spreadsheet cell have been exercised.

9. A method according to claim 8 wherein marking as exercised a subset of the test elements includes marking du-associations for the first spreadsheet cell that were exercised by the execution trace for the first spreadsheet cell, where each du-association includes a definition of a second cell and a use of the definition of the second cell in the formula for the first spreadsheet cell.

10. A method according to claim 9 wherein marking du-associations includes marking as exercised each du-association whose use is part of the formula for the first spreadsheet cell that has been executed.

11. A method according to claim 3 wherein marking as exercised a subset of the test elements includes:

receiving a user validation of the first spreadsheet cell; and

after receiving the user validation of the first spreadsheet cell, marking as exercised the subset of test elements that participated in the execution trace for the first spreadsheet cell.

12. A method according to claim 11 further comprising:

identifying all other spreadsheet cells upon which the first spreadsheet cell depends; and

for each other spreadsheet cell, repeating the steps of collecting, tracking, and marking so that any test elements for each other spreadsheet cell are marked as exercised independently of whether each other spreadsheet cell has been user-validated.

13. A method according to claim 11 wherein receiving a user validation includes having a user click on a validation tab for the first spreadsheet cell.

14. A method according to claim 11 wherein receiving a user validation includes:

storing a validation symbol for the first spreadsheet cell; and

displaying the validation symbol for the first spreadsheet cell.

15. A method according to claim 14 wherein displaying the validation symbol includes displaying the validation symbol for the first spreadsheet cell in a validation tab.

16. A method according to claim 3 further comprising:

discarding the test elements for the first spreadsheet cell after a user changes the formula for the first spreadsheet cell; and

repeating the steps of collecting, tracking, and marking for the first spreadsheet cell.

17. A method according to claim 16 wherein discarding the test elements for the first spreadsheet cell includes:

identifying all other cells in the spreadsheet that depend on the first spreadsheet cell; and

marking as unexercised all test elements for the identified cells that use a definition of the first spreadsheet cell.

18. A method according to claim 17 wherein marking as unexercised all test elements for the identified cells includes invalidating a user validation for the identified cells.

19. A method according to claim 18 wherein invalidating a user validation includes changing a validation symbol representing a validated spreadsheet cell for the identified cells to a validation symbol representing uncertainty for the identified cells.

20. A method according to claim 19 wherein invalidating a user validation includes displaying the validation symbol representing uncertainty for the identified cells.

21. A method according to claim 20 wherein displaying the changed validation symbol includes displaying the validation symbol representing uncertainty for the identified cells in a validation tab.

22. A method according to claim 19 wherein:

the validation symbol representing a validated spreadsheet cell is a checkmark; and

the validation symbol representing uncertainty is a question mark.

23. A method according to claim 18 wherein invalidating a user validation includes removing a validation symbol representing a validated spreadsheet cell for the identified cells.

24. A method according to claim 16 wherein discarding the test elements for the first spreadsheet cell includes invalidating a user validation for the first spreadsheet cell.

25. A method according to claim 24 wherein invalidating a user validation includes removing a validation symbol representing a validated spreadsheet cell for the first spreadsheet cell.

26. A method according to claim 25 wherein:
the validation symbol representing a validated spreadsheet cell is a checkmark.

27. A method according to claim 3 further comprising marking as unexercised a subset of the test elements for the first spreadsheet cell that are affected by a change in the formula for the first spreadsheet cell.

28. A method according to claim 3 further comprising providing the user with an indication of whether the first spreadsheet cell has been completely tested.

29. A method according to claim 3 further comprising providing the user with an indication of how completely the first spreadsheet cell has been tested.

30. A method according to claim 3 wherein the test elements are chosen from the group consisting of du-associations, node identifications, edge identifications, and cell identifications.

31. A method according to claim 3 further comprising selecting the first spreadsheet cell from the plurality of spreadsheet cells.

32. In a computerized spreadsheet containing a plurality of spreadsheet cells, a method for batch-collecting one or more test elements for a group of spreadsheet cells containing formulae to facilitate testing a first spreadsheet cell, the method comprising:

identifying all the spreadsheet cells upon which the group depends; and

collecting one or more test elements for each spreadsheet cell that depends upon an identified cell or the group of spreadsheet cells.

33. In a computerized spreadsheet, a method for providing a user with an indication of how completely tested a spreadsheet cell is, the method comprising:

maintaining a set of test elements for the spreadsheet cell;

identifying a subset of the set of test elements that have been exercised;

using the set of test elements and the subset of the set, calculating a testedness measure of test elements that have been exercised; and

providing the user with an indication of the testedness measure.

34. A method according to claim 33 wherein providing the user with an indication of the testedness measure includes providing the user with visual feedback of the testedness measure.

35. A method according to claim 34 wherein providing the user with visual feedback includes coloring a border of the spreadsheet cell, where the color chosen indicates the testedness measure.

36. A method according to claim 35 wherein the color chosen to indicate the testedness measure varies between red and blue, the color red representing a 0% testedness percentage and the color blue representing a 100% testedness percentage.

37. A method according to claim 33 further including updating the indication of the testedness measure as the testedness measure changes.

38. In a computerized spreadsheet, a method for providing a user with an indication of how completely tested a spreadsheet cell is, the method comprising:

maintaining a test element for the spreadsheet cell;

calculating a testedness measure of the test element for the spreadsheet cell; and

providing the user with an indication of the testedness measure of the test element for the spreadsheet cell.

39. A method according to claim 38 wherein providing the user with an indication of the testedness measure of the test element for the spreadsheet cell includes providing the user with visual feedback of the testedness measure of the test element for the spreadsheet cell.

40. A method according to claim 39 wherein providing the user with visual feedback of the testedness measure of the test element for the spreadsheet cell includes coloring an arrow representing the test element for the spreadsheet cell, where the color chosen indicates the testedness measure.

41. A method according to claim 40 wherein the color chosen to indicate the testedness measure varies between red and blue, the color red representing a 0% testedness percentage and the color blue representing a 100% testedness percentage.

42. A method according to claim 38 further including updating the indication of the testedness measure as the testedness measure changes.

43. A computer-readable medium containing a testing program to support testing, in a computerized spreadsheet, a spreadsheet cell containing a formula, the testing program comprising:

a test element data structure for storing a set of one or more test elements for the spreadsheet cell;

a trace data structure for storing an execution trace of the spreadsheet cell; and

a subroutine operable to software mark as exercised any test elements that participated in the execution trace.

44. A testing program according to claim 43 further comprising a subroutine operable to update the test element data structure as the formula for the spreadsheet cell changes.

45. A testing program according to claim 43 further comprising a subroutine operable to update the trace data structure as any cell upon which the spreadsheet cell depends changes.

46. A testing program according to claim 43 further comprising:

a subroutine operable to identify all other spreadsheet cells upon which the spreadsheet cell depends; and

a subroutine operable to update the test element data structure for each other spreadsheet cell.

47. A testing program according to claim 43 wherein each test element includes:

a definition reference to a definition of a second cell;

a use reference to a use of the definition reference of the second cell in the formula of the spreadsheet cell; and

a du-association between the definition reference and the use reference.

48. A testing program according to claim 43 further comprising a validation structure that allows a user to mark the spreadsheet cell as validated.

49. A testing program according to claim 48 wherein the validation structure changes from a checkmark to a question mark when the formula of the spreadsheet cell changes.

50. A testing program according to claim 48 wherein the validation structure changes from a checkmark to a blank when the formula of the spreadsheet cell changes.

51. A system for testing a spreadsheet cell in a spreadsheet on a computer system, the system comprising:

a set of one or more test elements for the spreadsheet cell;

an execution trace of the spreadsheet cell; and

a marking unit to mark as exercised test elements that participated in the execution trace.

52. A system according to claim 51 further comprising:

a dependency identification unit to identify all other spreadsheet cells upon which the spreadsheet cell depends; and

a test element update unit to update the test element data structure for each other spreadsheet cell.

53. A system according to claim 51 wherein each test element includes:

a definition reference to a definition of a second cell;

a use reference to a use of the definition reference of the second cell in the formula of the spreadsheet cell; and

a du-association between the definition reference and the use reference.

54. A system according to claim 51 further comprising a validation structure that allows a user to mark the spreadsheet cell as validated.

55. A system according to claim 54 wherein the validation structure is constructed and arranged to change from a checkmark to a question mark when the formula of the spreadsheet cell changes.

## BIBLIOGRAPHY

(Aho86) A. V. Aho, R. Sethi, and J. D. Ullman. Compilers, Principles, Techniques, and Tools. Addison-Wesley Publishing Company, Reading, MA, 1986.

(Ambler92) A. Ambler, M. Burnett, and B. Zimmerman. Operational versus definitional: a perspective on programming paradigms. Computer, 25(9):28-43, September 1992.

(Azem93) A. Azem, F. Belli, O. Jack, and P. Jedrzejowicz. Testing and reliability of logic programs. In The Fourth International Symposium on Software Reliability Engineering, pages 318-327, 1993.

(Belli95) F. Belli and O. Jack. A test coverage notion for logic programming. In The 6th International Symposium on Software Reliability Engineering, pages 133-142, 1995.

(Brown87) P. Brown and J. Gould. Experimental study of people creating spreadsheets. ACM Transactions on Office Information Systems, 5(3):258-272, July 1987.

(Burnett98a) M. Burnett, J. Atwood, and Z. Welch. Implementing level 4 liveness in declarative visual programming languages. In 1998 IEEE Symposium on Visual Languages, September 1998.

(Burnett98b) M. Burnett and H. Gottfried. Graphical definitions: Expanding spreadsheet languages through direct manipulation and gestures. ACM Transactions on Computer-Human Interaction, 5(1):1-33, March 1998.

(Burnett94) M. Burnett, R. Hossli, T. Pulliam, B. VanVoorst, and X. Yang. Toward visual programming languages for steering in scientific visualization: a taxonomy. IEEE Comp. Science and Engineering, 1(4), 1994.

(Chi97) E. H. Chi, P. Barry, J. Riedl, and J. Konstan. A Spreadsheet Approach to Information Visualization. In IEEE Symposium on Information Visualization, October 1997.

(Christ75) R. Christ. Review and analysis of color coding research for visual displays. Human Factors, 17(6):542-570, 1975.

(Clarke76) L. Clarke. A system to generate test data and symbolically execute programs. IEEE Transactions on Software Engineering, 2(3), September 1976.

(Clarke89) L Clarke, A. Podgurski, D. Richardson, and S. Zeil. A formal evaluation of data flow path selection criteria. IEEE Transactions on Software Engineering, SE-15(11):1318-1332, November 1989.

(Cook99) C. Cook, K. Rothermel, M. Burnett, T. Adams, G. Rothermel, A. Sheretov, F. Cort, J. Reichwein. Does immediate visual feedback about testing aid debugging in spreadsheet language? TR 99-60-07, Oregon State University, March 1999.

(Duesterwald92) E. Duesterwald, R. Gupta, and M. L. Soffa. Rigorous data flow testing through output influences. In Proceedings of the 2nd Irvine Software Symposium, March 1992.

(Frankl93) P. Frankl and S. Weiss. An experimental comparison of the effectiveness of branch testing and data flow testing. IEEE Transactions on Software Engineering, 19(8):774-787, August 1993.

(Frankl88) P. Frankl and E. Weyuker. An applicable family of data flow criteria. IEEE Transactions on Software Engineering, 14(10):1483-1498, October 1988.

(Green96) T. Green and M. Petre. Usability analysis of visual programming environments: A 'cognitive dimensions' framework. Journal of Visual Languages and Computing, 7(2):131-174, June 1996.

(Gupta93) R. Gupta, M. J. Harrold, and M. L. Soffa. Program slicing-based regression testing techniques. Journal of Software Testing, Verification, and Reliability, 6(2):83-112, June 1996.

(Harrold88) M. J. Harrold and M. L. Soffa. An incremental approach to unit testing during maintenance. In Proceedings of the Conference on Software Maintenance, pages 362-367, October 1988.

(Hudson91) S. Hudson. Incremental attribute evaluation: a flexible algorithm for lazy update. ACM Transactions Programming Languages and Systems, 13(3), July 1991.

(Hutchins94) M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments on the effectiveness of dataflow- and controlflow-based test adequacy criteria. In 16th International Conference on Software Engineering, pages 191-200, May 1994.

(Kuhn97) W. Kuhn and A. U. Frank. The use of functional programming in the specification and testing process. In International Conference and Workshop on Interoperating Geographic Information Systems, December 1997.

(Laski83) J. Laski and B. Korel. A data flow oriented program testing strategy. IEEE Transactions on Software Engineering, 9(3):347-354, May 1993.

(Leopold97) J. Leopold and A. Ambler. Keyboardless visual programming using voice, handwriting, and gesture. In 1997 IEEE Symposium on Visual Languages, pages 28-35, September 1997.

(Luo92) G. Luo, G. Bochmann, B. Sarikaya, and M. Boyer. Control-flow based testing of Prolog programs. In The 3rd International Symposium on Software Reliability Engineering, pages 104-113, 1992.

(Marlowe90) T. Marlowe and B. Ryder. An efficient hybrid algorithm for incremental data flow analysis. In ACM Principles of Programming Languages, pages 184-196, January 1990.

(Murch84) G. Murch. Physiological principles for the effective use of color. IEEE Computer Graphics and Applications, pages 49-54, November 1984.

(Myers91) B. Myers. Graphical techniques in a spreadsheet for specifying user interfaces. In ACM Conference on Human Factors in Computing Systems, pages 243-249, April 1991.

(Ntafos84) S. C. Ntafos. On required element testing. IEEE Transactions on Software Engineering, 10(6), November 1984.

(Offutt96) J. Offutt, J. Pan, K. Tewary, and T. Zhang. An experimental evaluation of data flow and mutation testing. Software Practice and Experience, 26(2):165-176, February 1996.

(Ouabd95) F. Ouabdesselam and I. Parissis. Testing techniques for data-flow synchronous programs. In AADEBUG'95: Second International Workshop on Automated and Algorithmic Debugging, May 1995.

(Panko96) R. Panko and R. Halverson. Spreadsheets on trial: A survey of research on spreadsheet risks. In Twenty-Ninth Hawaii International Conference on System Sciences, January 1996.

(Perry90) D. Perry and G. Kaiser. Adequate testing and object-oriented programming. Journal of Object-Oriented Programming, 2, January 1990.

(Pollock89) L. Pollock and M. L. Soffa. An incremental version of iterative data flow analysis. IEEE Transactions on Software Engineering, 15(12):1537-1549, December 1989.

(Rapps85) S. Rapps and E. J. Weyuker. Selecting software test data using data flow information. IEEE Transactions on Software Engineering, 11(4):367-375, April 1985.

(Rothermel94) G. Rothermel and M. J. Harrold. Selecting tests and identifying test coverage requirements for modified software. In Proceedings of the 1994 International Symposium Software Testing and Analysis, pages 169-184, August 1994.

(Rothermel97a) G. Rothermel and M.J. Harrold. A safe, efficient regression test selection technique. ACM Transactions on Software Engineering and Methodology, 6(2):173-210, April 1997.

(Rothermel97b) G. Rothermel, L. Li, and M. Burnett. Testing strategies for form-based visual programs. In The Eighth International Symposium on Software Reliability Engineering, pages 96-107, November 1997.

(Shneiderman98) B. Shneiderman. Designing the User Interface. Addison-Wesley, Reading, MA, 3rd edition, 1998.

(Smedley96) T. Smedley, P. Cox, and S. Byrne. Expanding the utility of spreadsheets through the integration of visual programming and user interface objects. In Advanced Visual Interfaces '96, May 1996.

(Viehstaedt92) G. Viehstaedt and A. Ambler. Visual representation and manipulation of matrices. Journal of Visual Languages and Computing, 3(3):273-298, September 1992.

(Weyuker86) E. J. Weyuker. Axiomatizing software test data adequacy. IEEE Transactions on Software Engineering, 12(12):1128-1138, December 1986.

(Weyuker93) E. J. Weyuker. More experience with dataflow testing. IEEE Transactions on Software Engineering, 19(9), September 1993.

(Wilcox97) E. Wilcox, J. Atwood, M. Burnett, J. Cadiz, and C. Cook. Does continuous visual feedback aid debugging in direct-manipulation programming systems? In ACM Conference on Human Factors in Computing Systems, pages 22-27, March 1997.

(Wong95) W. Wong, R. Horgan, S. London, and A. Mathur. Effect of test set minimization on fault detection effectiveness. In 17th International Conference on Software Engineering, pages 41-50, April 1995.

(Yang97) S. Yang, M. Burnett, E. DeKoven, and M. Zloof. Representation design benchmarks: A design-time aid for VPL navigable static representations. Journal of Visual Languages and Computing, 8(5/6):563-599, Oct/Dec 1997.

6954-2.app.ps.pdf

## METHODOLOGY FOR TESTING SPREADSHEETS

## ABSTRACT OF THE DISCLOSURE

The invention includes a method for testing a spreadsheet cell. Du-associations are collected for the spreadsheet cell. The cell's execution trace is tracked. After the user validates the cell, the du-associations that participated in the execution trace are marked as executed. Du-associations for other cells that are affected by the testing a spreadsheet cell are similarly marking as exercised.

If the user changes a cell's contents, the collected du-associations for the cell are discarded and the steps of collecting test elements, tracking execution traces, and marking are repeated. Cells that depend on the changed cell also repeat these steps.

After the user marks a cell as validated, a validation symbol is shown on the cell. If the cell's validation status is later brought into question, the validation symbol can change or be removed entirely.

The invention also includes a method for providing a user with feedback of the testedness of the spreadsheet cells. The du-associations for each spreadsheet cell are maintained, and the subset of du-associations that have been exercised is identified. Using the numbers of tested and untested du-associations, a testedness measure is calculated for the cell, which is provided to the user.

Client's Docket No. 6954

660777 "80886060

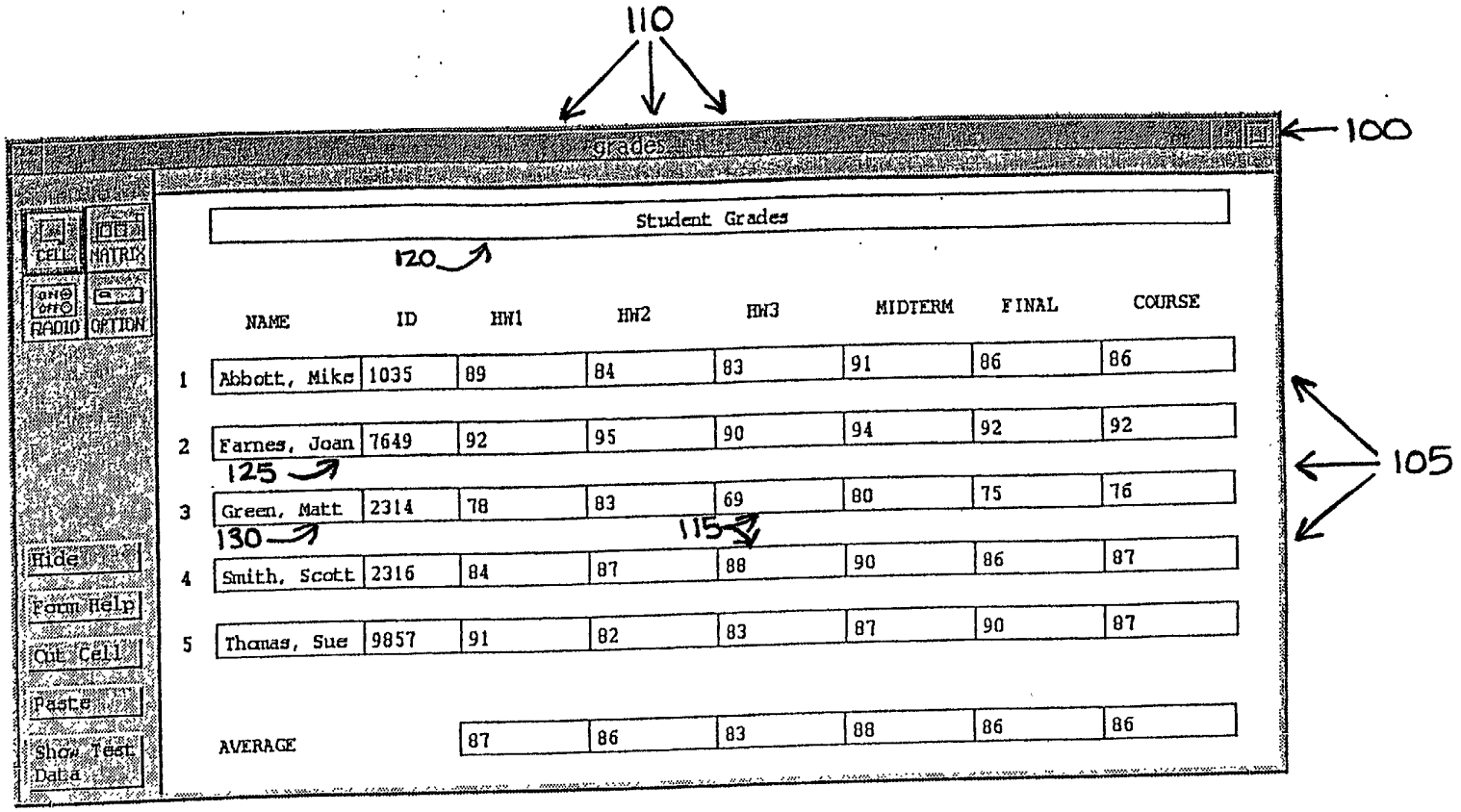


FIG. 1

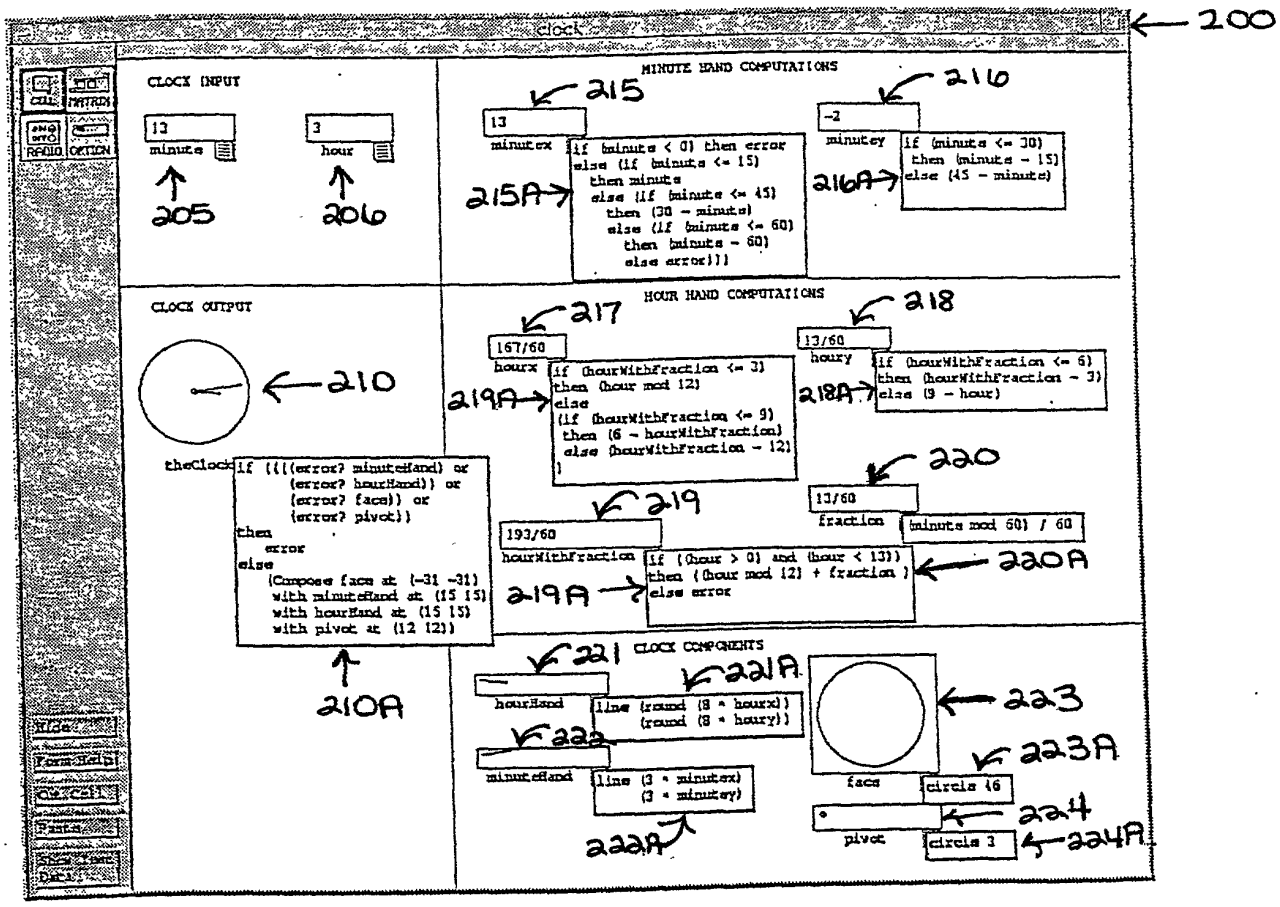
clock
200
CLOCK INPUT
13
minute
205
3
hour
206
MINUTE HAND COMPUTATIONS
215
13
minutex
215A
if (minute < 0) then error
else (if (minute <= 15)
then minute
else (if (minute <= 45)
then (30 - minute)
else (if (minute <= 60)
then (minute - 60)
else error)))
216
-2
minutey
216A
if (minute <= 30)
then (minute - 15)
else (45 - minute)
CLOCK OUTPUT
210
theClock
if ((((error? minuteHand) or
(error? hourHand)) or
(error? face)) or
(error? pivot))
then
error
else
(Compose face at (-31 -31)
with minuteHand at (15 15)
with hourHand at (15 15)
with pivot at (12 12))
210A
HOUR HAND COMPUTATIONS
217
167/60
hourx
219A
if (hourWithFraction <= 3)
then (hour mod 12)
else
(if (hourWithFraction <= 9)
then (6 - hourWithFraction)
else (hourWithFraction - 12)
)
218
13/60
houry
218A
if (hourWithFraction <= 6)
then (hourWithFraction - 3)
else (9 - hour)
219
193/60
hourWithFraction
219A
220
13/60
fraction
minute mod 60) / 60
if (hour > 0) and (hour < 13))
then ((hour mod 12) + fraction )
else error
220A
CLOCK COMPONENTS
221
hourHand
221A
line (round (8 * hourx))
(round (8 * houry))
222
minuteHand
line (3 * minutex)
(3 * minutey)
222A
223
face
circle 46
223A
224
pivot
circle 3
224A

FIG. 2

5607674 "4502516

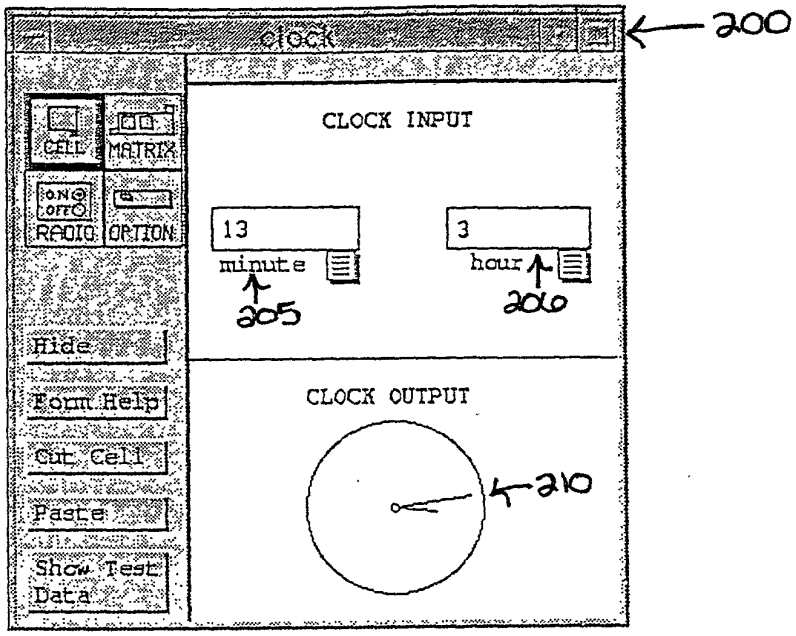


FIG. 3

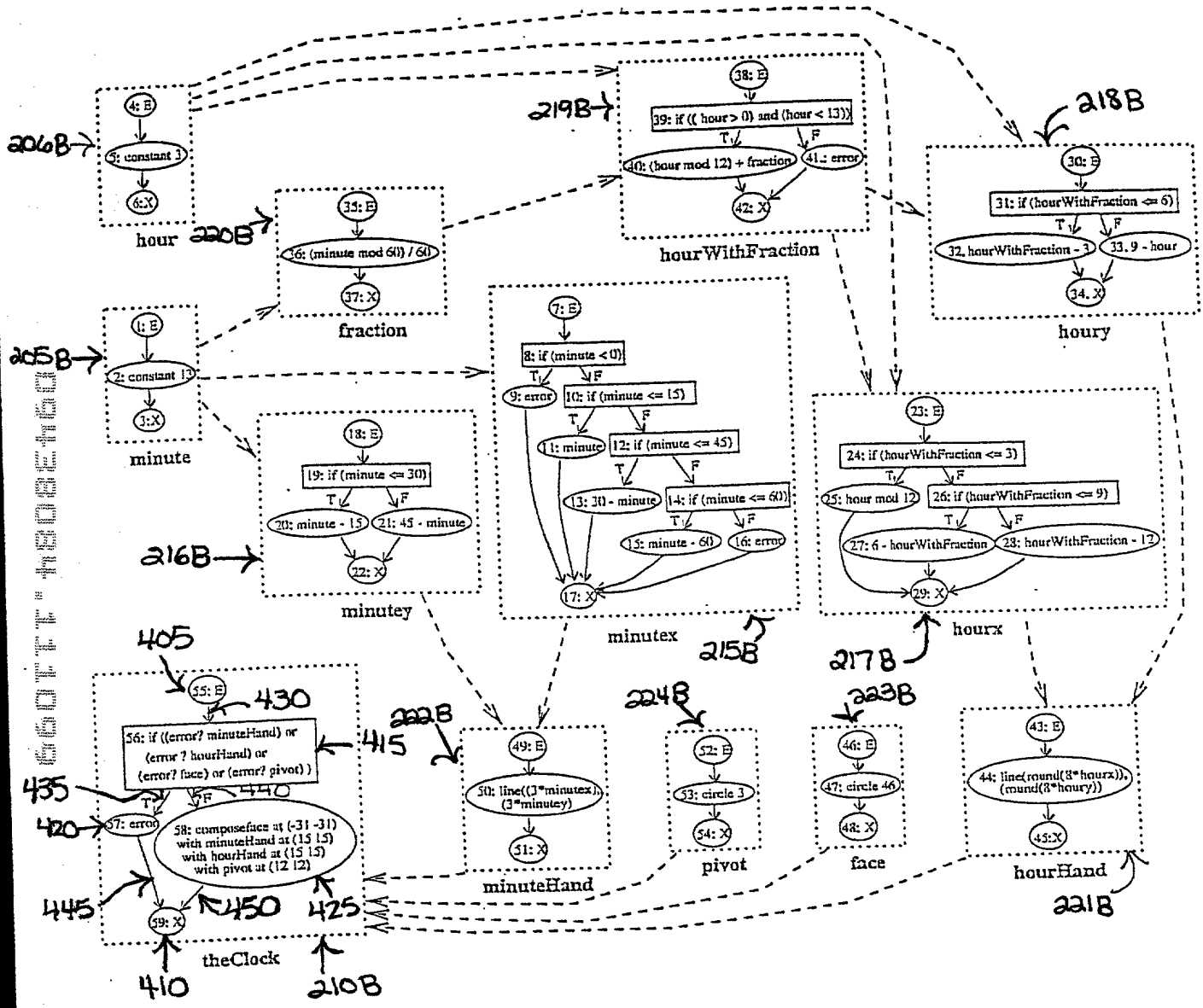
4: E
5: constant 3
6: X
hour
206B
35: E
36: (minute mod 60) / 60
37: X
fraction
220B
38: E
39: if (( hour > 0) and (hour < 13))
T
F
40: (hour mod 12) + fraction
41.: error
42: X
hourWithFraction
219B
30: E
31: if (hourWithFraction <= 6)
T
F
32. hourWithFraction - 3
33. 9 - hour
34. X
houry
218B
1: E
2: constant 15
3: X
minute
205B
18: E
19: if (minute <= 30)
T
F
20: minute - 15
21: 45 - minute
22: X
minutey
216B
7: E
8: if (minute < 0)
T
F
9: error
10: if (minute <= 15)
T
F
11: minute
12: if (minute <= 45)
T
F
13: 30 - minute
14: if (minute <= 60)
T
F
15: minute - 60
16: error
17: X
minutex
215B
23: E
24: if (hourWithFraction <= 3)
T
F
25: hour mod 12
26: if (hourWithFraction <= 9)
T
F
27: 6 - hourWithFraction
28: hourWithFraction - 12
29: X
hourx
217B
405
55: E
430
56: if ((error? minuteHand) or (error ? hourHand) or (error? face) or (error? pivot) )
T
F
440
415
435
420
57: error
58: composeface at (-31 -31) with minuteHand at (15 15) with hourHand at (15 15) with pivot at (12 12)
445
450
425
59: X
theClock
410
210B
222B
49: E
50: line((3*minutex), (3*minutey)
51: X
minuteHand
224B
52: E
53: circle 3
54: X
pivot
223B
46: E
47: circle 46
48: X
face
43: E
44: line(round(8*hourx)), (round(8*houry))
45: X
hourHand
221B

FIG. 4

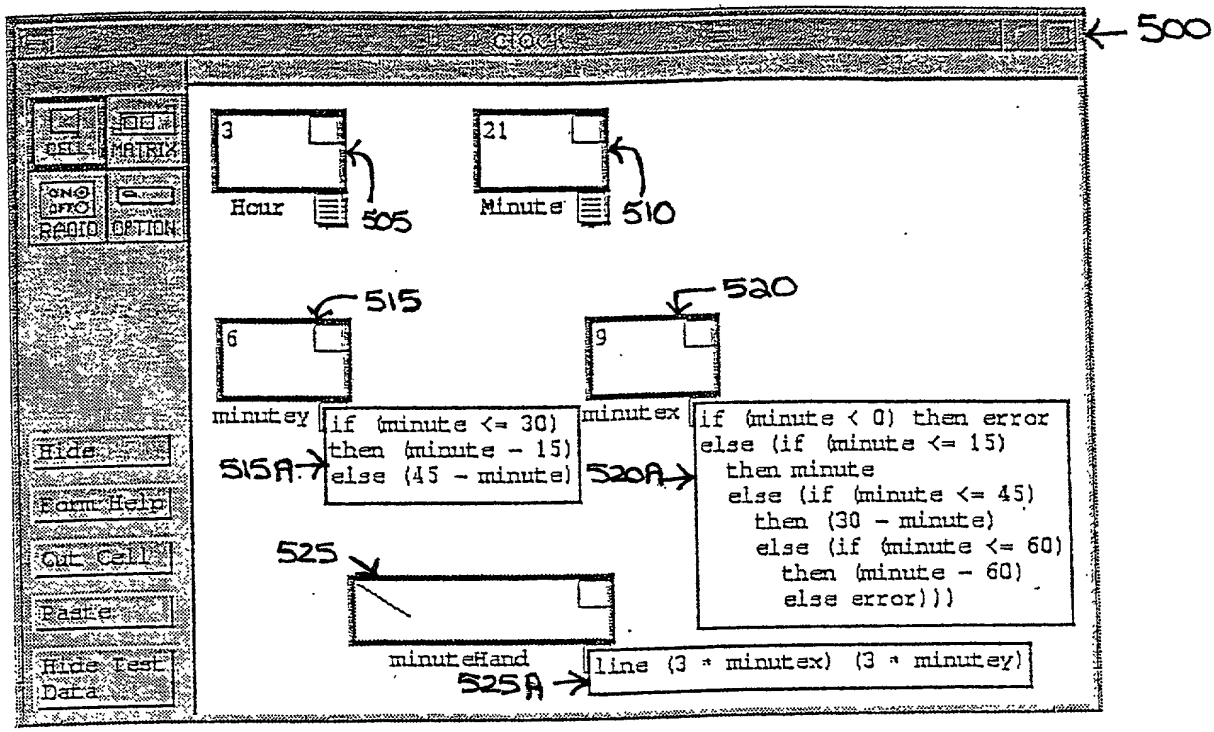
500
Clock
OFF
MATRIX
RADIO
OPTION
Hide
Form Help
Cut Cell
Paste
Hide Test Data
3
Hour
505
21
Minute
510
515
6
minutey
if (minute <= 30)
then (minute - 15)
else (45 - minute)
515A
520
9
minutex
if (minute < 0) then error
else (if (minute <= 15)
then minute
else (if (minute <= 45)
then (30 - minute)
else (if (minute <= 60)
then (minute - 60)
else error)))
520A
525
minuteHand
line (3 * minutex) (3 * minutey)
525A

FIG. 5

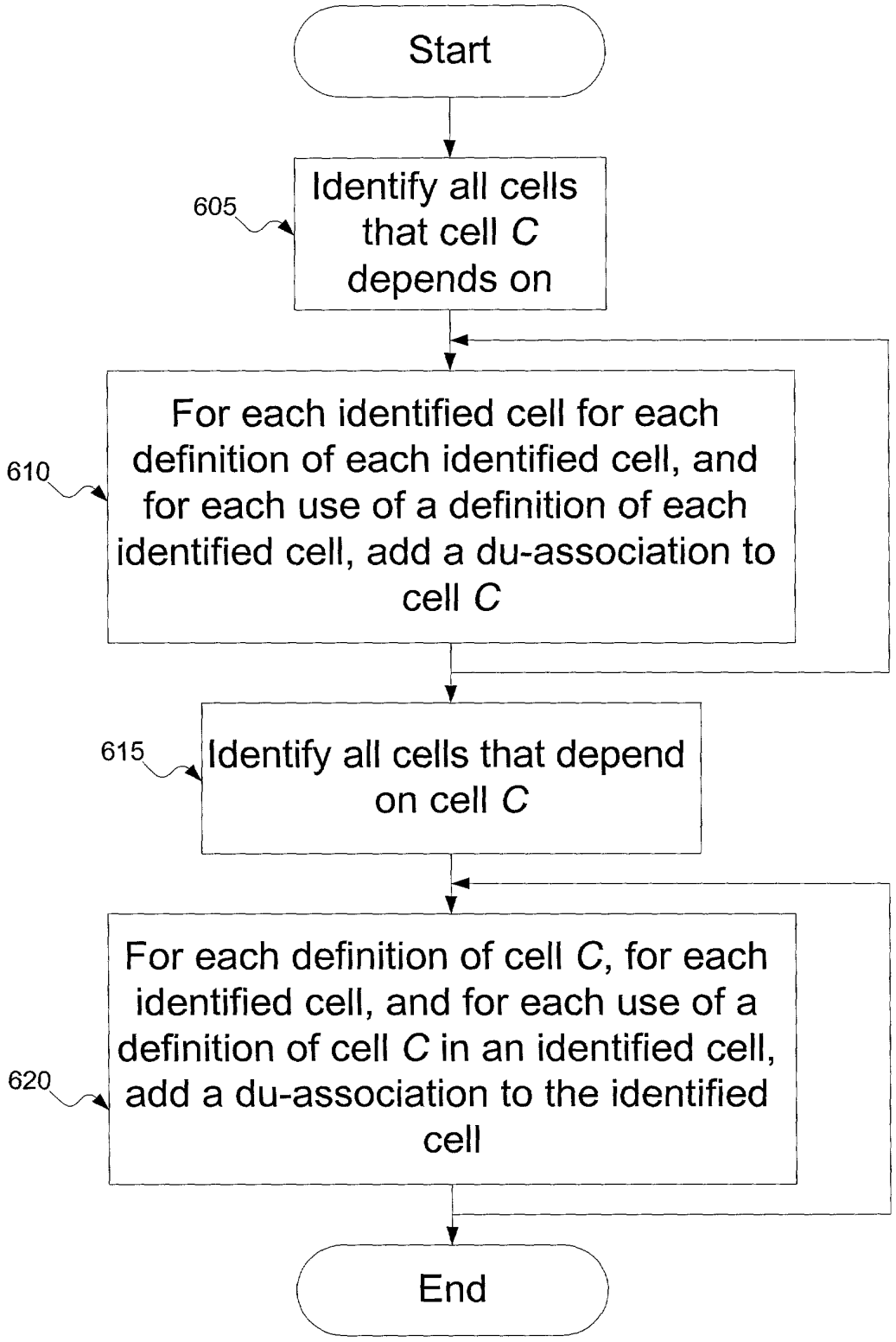
Start
605
Identify all cells that cell C depends on
610
For each identified cell for each definition of each identified cell, and for each use of a definition of each identified cell, add a du-association to cell C
615
Identify all cells that depend on cell C
620
For each definition of cell C, for each identified cell, and for each use of a definition of cell C in an identified cell, add a du-association to the identified cell
End

FIG. 6

660TTT"h808Eh60

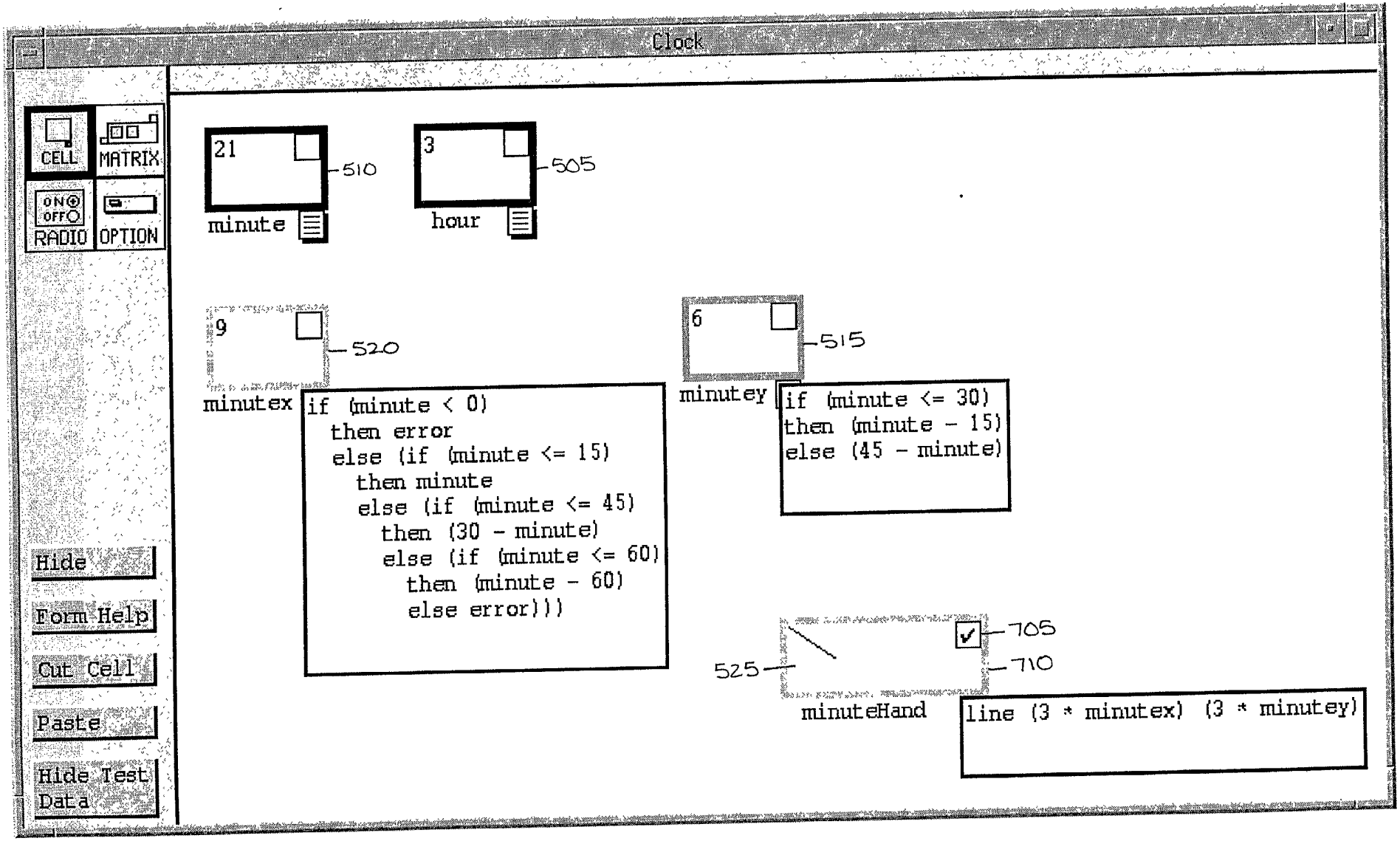


FIG. 7

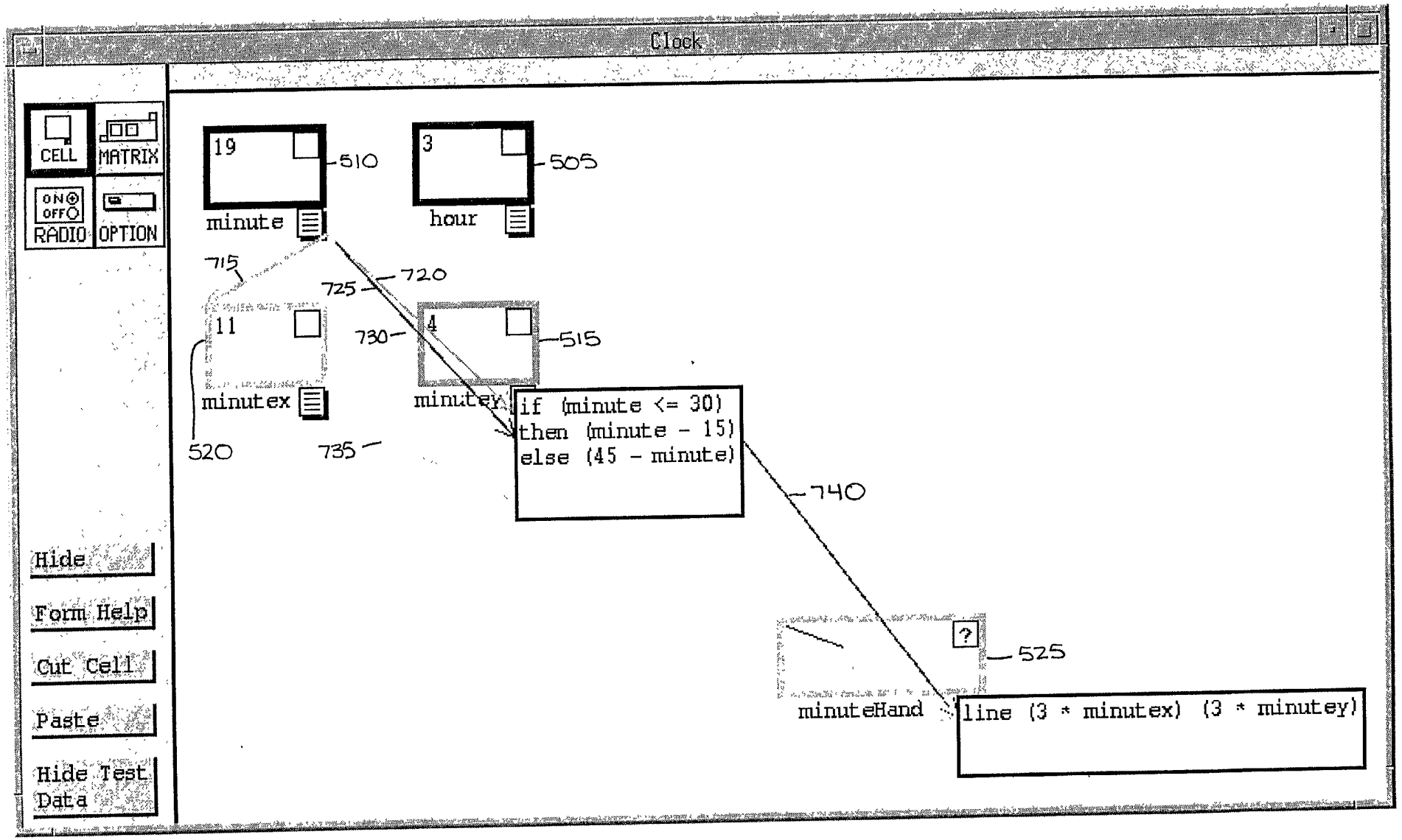
Clock
CELL
MATRIX
ON
OFF
RADIO
OPTION
19
510
minute
3
505
hour
715
725
720
11
730
4
515
minutex
minutey
520
735
if (minute <= 30)
then (minute - 15)
else (45 - minute)
740
?
525
minuteHand
line (3 * minutex) (3 * minutey)
Hide
Form Help
Cut Cell
Paste
Hide Test Data

FIG. 7A

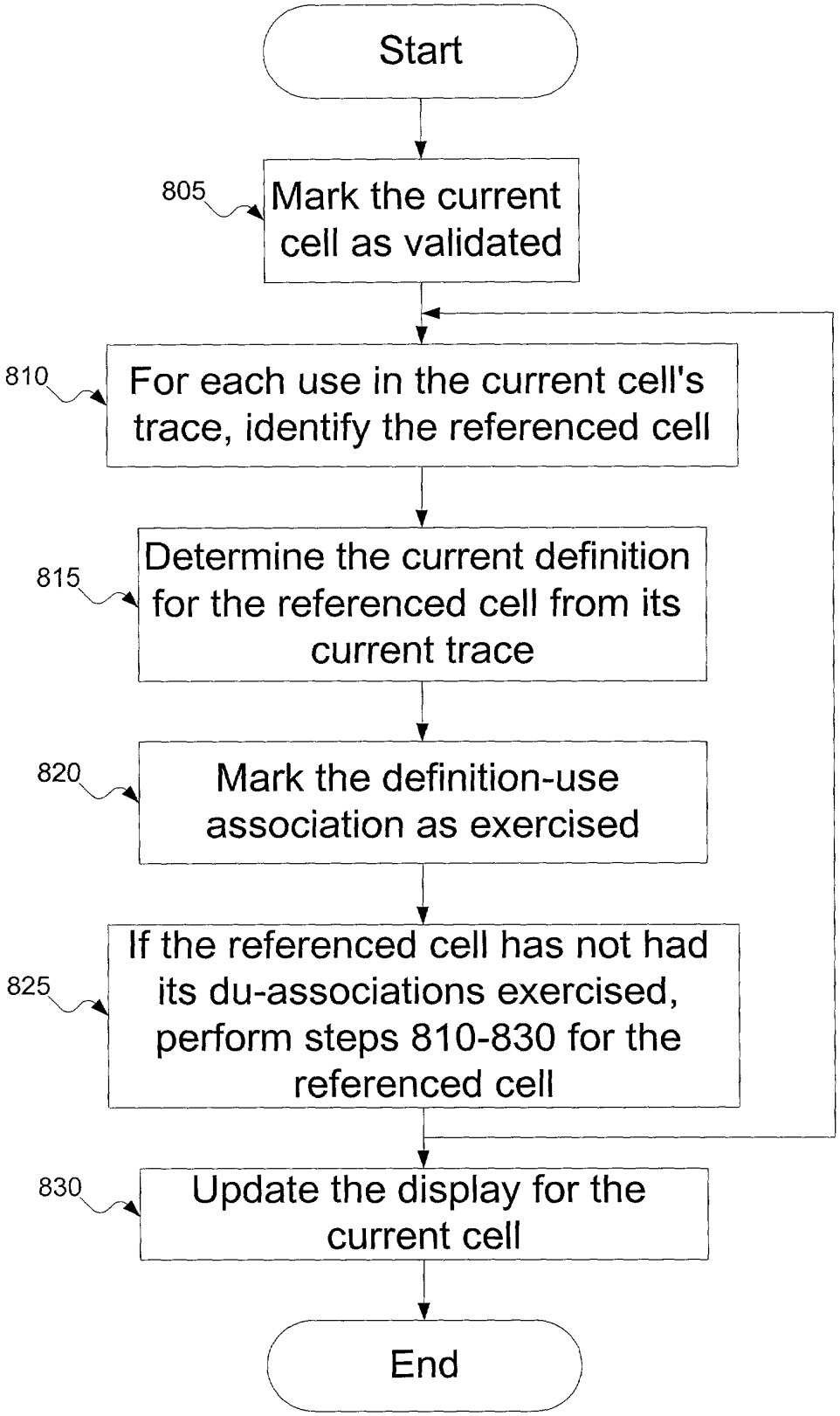
Start
805
Mark the current cell as validated
810
For each use in the current cell's trace, identify the referenced cell
815
Determine the current definition for the referenced cell from its current trace
820
Mark the definition-use association as exercised
825
If the referenced cell has not had its du-associations exercised, perform steps 810-830 for the referenced cell
830
Update the display for the current cell
End

FIG. 8

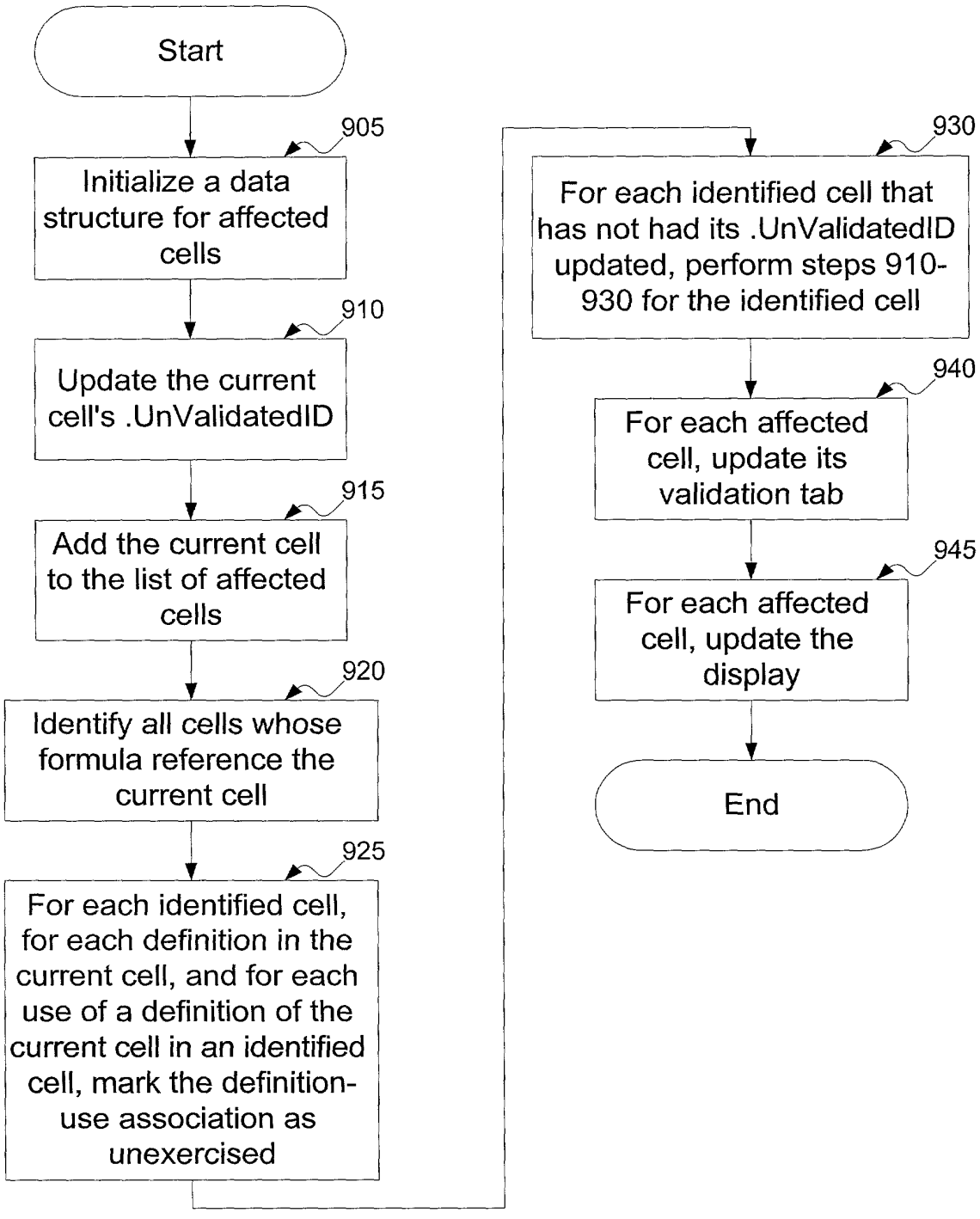
Start
905
Initialize a data structure for affected cells
910
Update the current cell's .UnValidatedID
915
Add the current cell to the list of affected cells
920
Identify all cells whose formula reference the current cell
925
For each identified cell, for each definition in the current cell, and for each use of a definition of the current cell in an identified cell, mark the definition-use association as unexercised
930
For each identified cell that has not had its .UnValidatedID updated, perform steps 910-930 for the identified cell
940
For each affected cell, update its validation tab
945
For each affected cell, update the display
End

FIG. 9

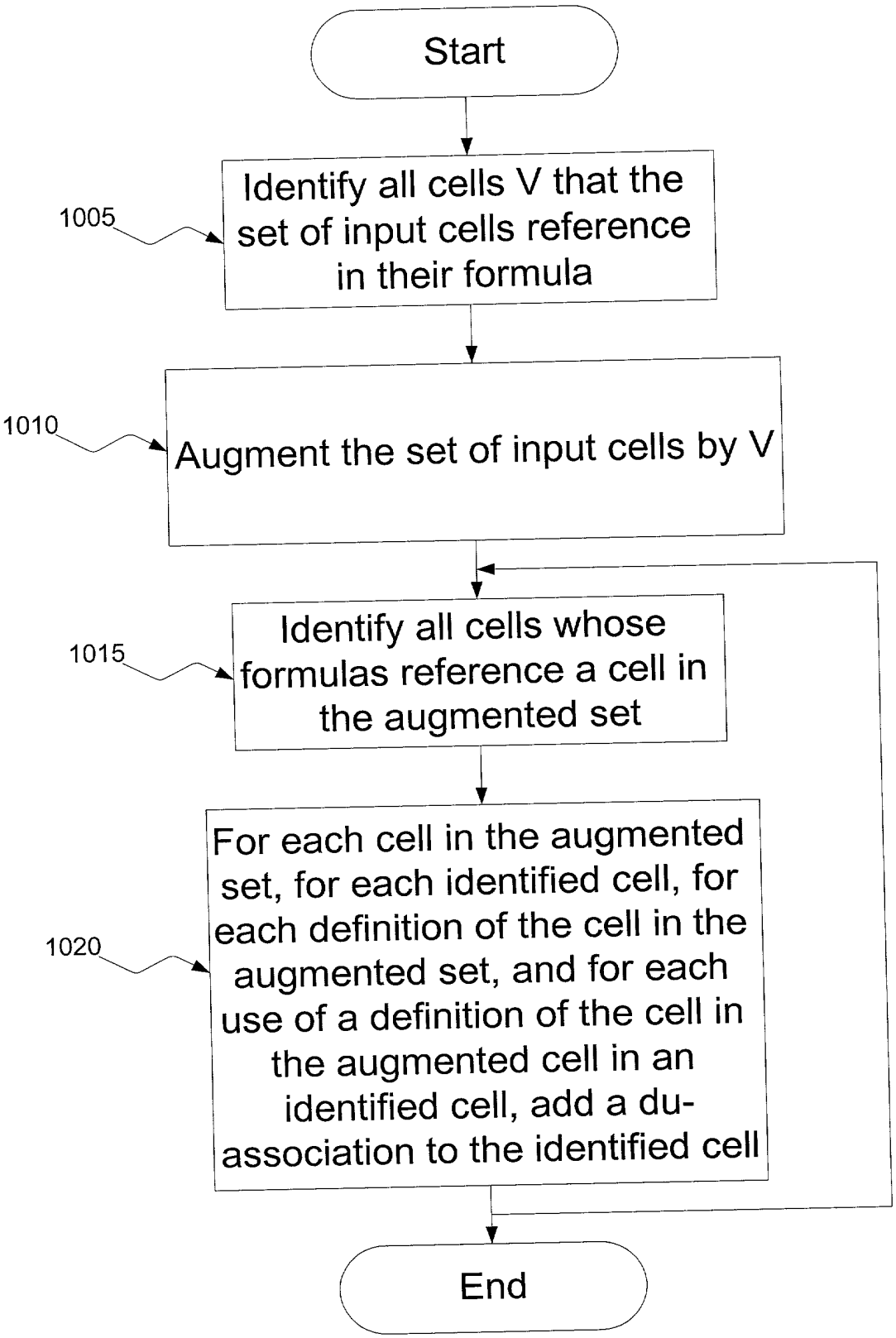
Start
Identify all cells V that the set of input cells reference in their formula
1005
Augment the set of input cells by V
1010
Identify all cells whose formulas reference a cell in the augmented set
1015
For each cell in the augmented set, for each identified cell, for each definition of the cell in the augmented set, and for each use of a definition of the cell in the augmented cell in an identified cell, add a du-association to the identified cell
1020
End

FIG. 10

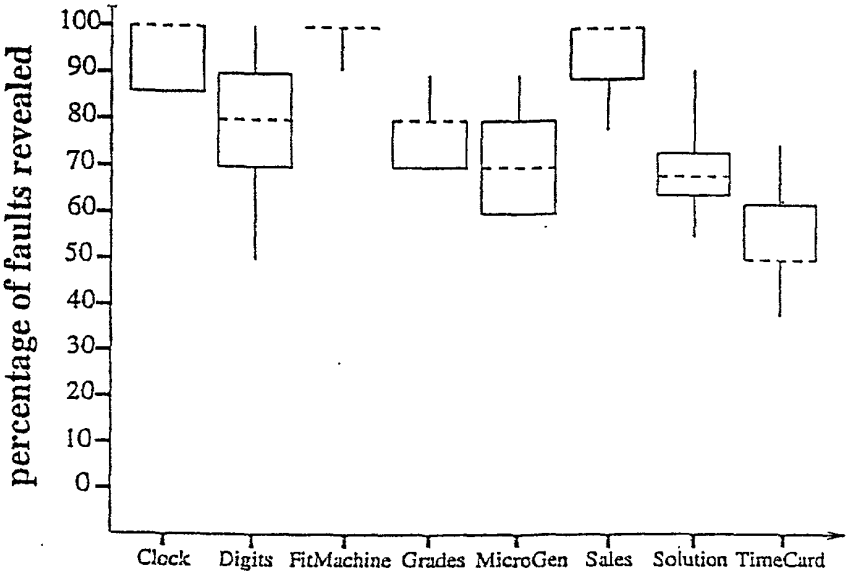
percentage of faults revealed
100
90
80
70
60
50
40
30
20
10
0
Clock
Digits
FitMachine
Grades
MicroGen
Sales
Solution
TimeCard

FIG. 11

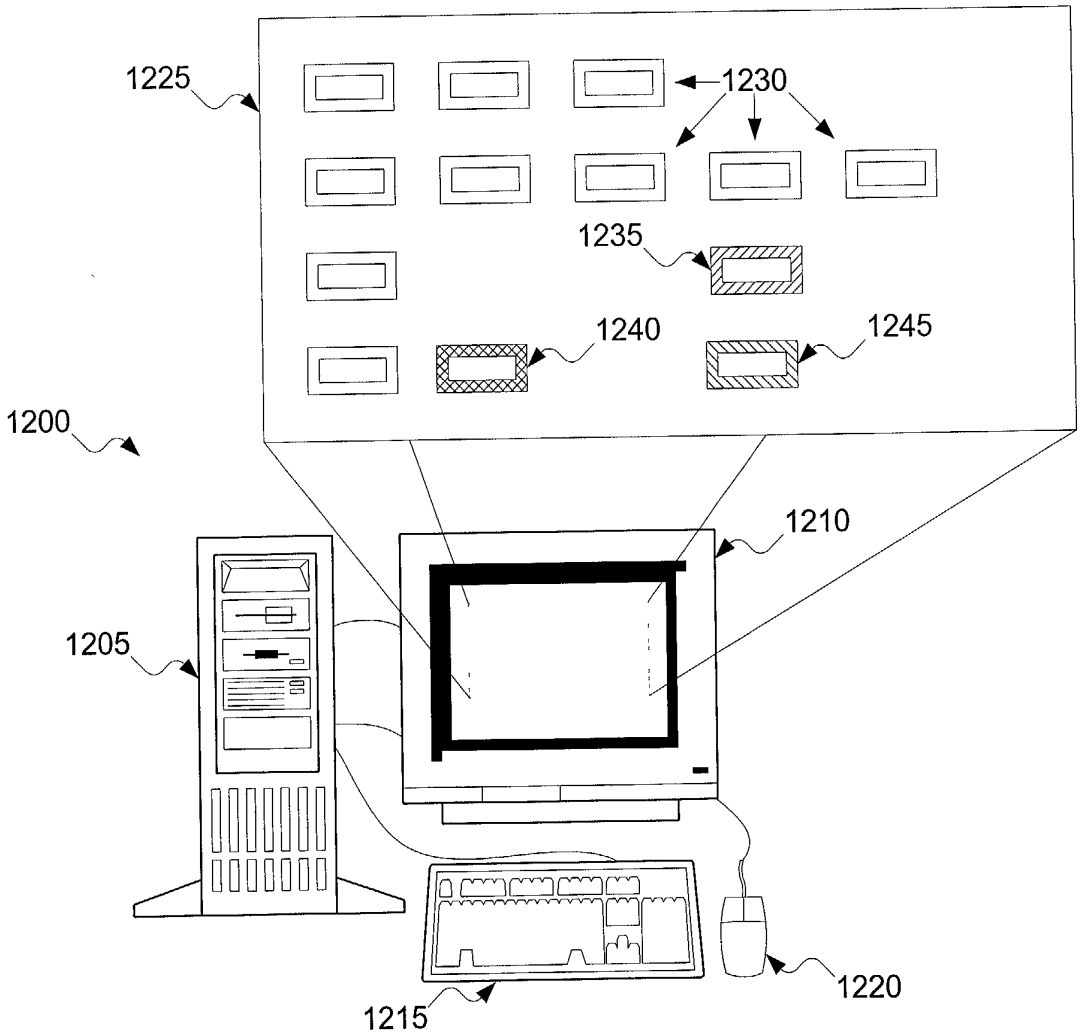


FIG. 12

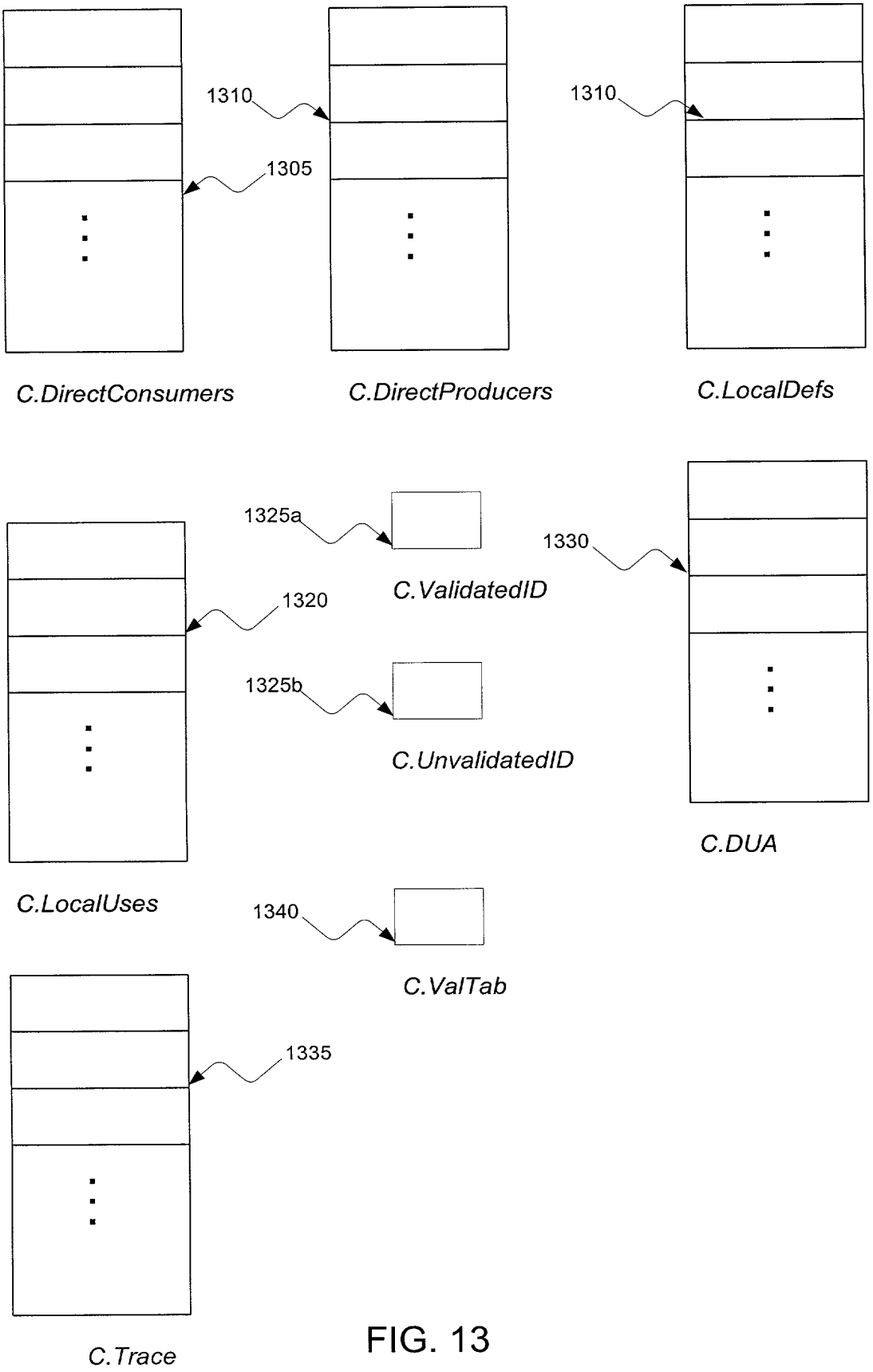
1310
1305
1310
C.DirectConsumers
C.DirectProducers
C.LocalDefs
1325a
C.ValidatedID
1320
1330
1325b
C.UnvalidatedID
C.DUA
C.LocalUses
1340
C.ValTab
1335
C.Trace

FIG. 13

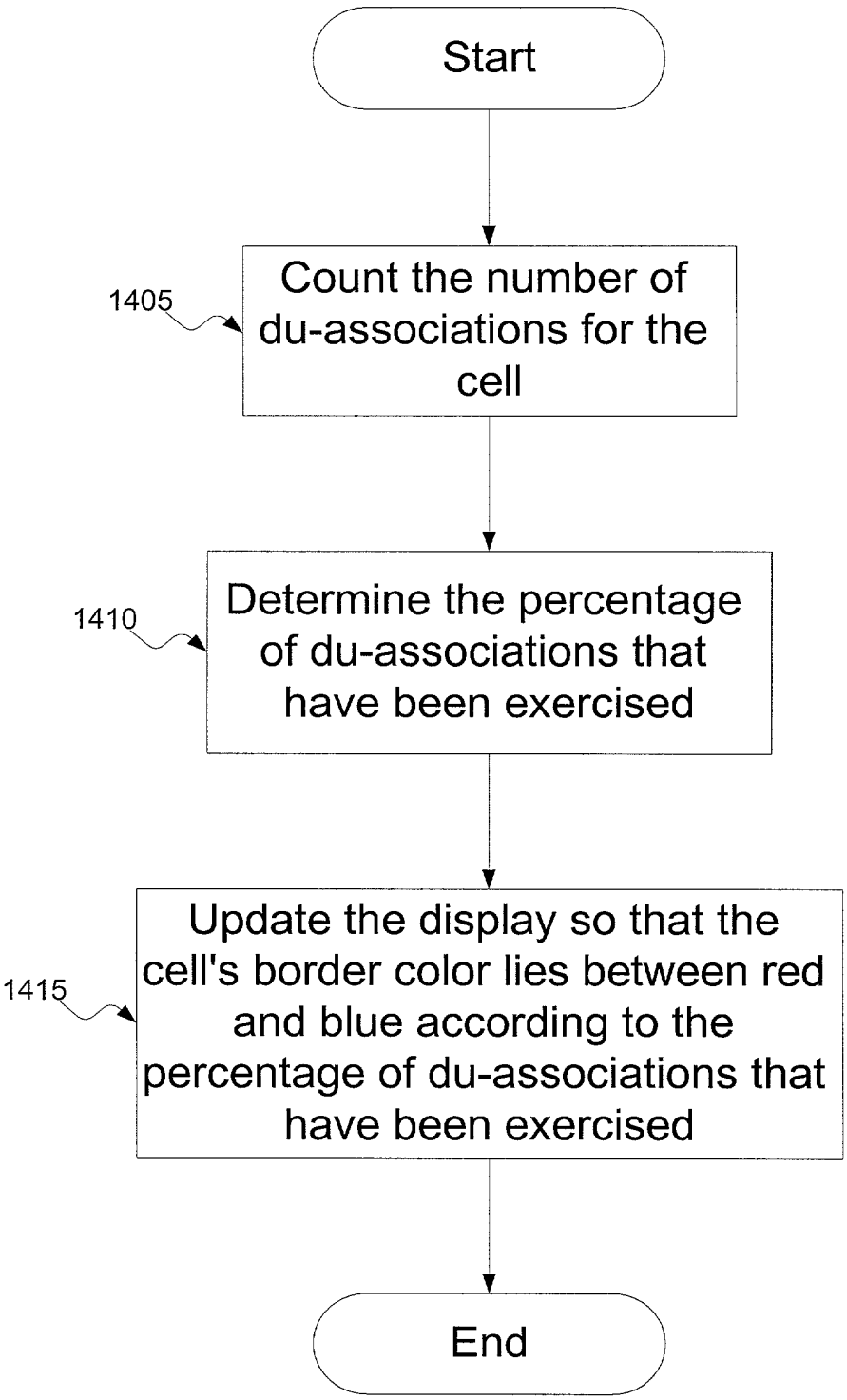
Start
1405
Count the number of du-associations for the cell
1410
Determine the percentage of du-associations that have been exercised
1415
Update the display so that the cell's border color lies between red and blue according to the percentage of du-associations that have been exercised
End

FIG. 14

Attorney's Do. No. 6954-2

# DECLARATION FOR PATENT APPLICATION

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name.

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled METHODOLOGY FOR TESTING SPREADSHEETS, the specification of which:

[X] is attached hereto.

[ ] was filed on _____ as Application Serial No. _____

[ ] and was amended on _____ (if applicable)

[ ] with amendments through _____ (if applicable)

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to patentability as defined in Title 37, Code of Federal Regulations, Sec. 1.56.

I hereby claim foreign priority benefits under Title 35, United States Code, Sec. 119(a)-(d) of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

| Prior Foreign Application(s) | | | Priority Claimed | |
|---|---|---|---|---|
| _____ (Number) | _____ (Country) | _____ (Day/Month/Year Filed) | [ ] Yes | [ ] No |

I hereby claim the benefit under Title 35, United States Code, Sec. 119(e) of any United States provisional application listed below:

| Provisional Application No. | Filing Date |
|---|---|
| 60/125,414 | March 22, 1999 |

I hereby claim the benefit under Title 35, United States Code, Sec. 120 of any United States application(s), or Sec. 365(c) of any PCT International application designating the United States, listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States or PCT International application in the manner provided by the first paragraph of Title 35, United States Code, Sec. 112. I acknowledge the duty to disclose information which is material to patentability as defined in Title 37, Code of Federal Regulations, Sec. 1.56 which became available between the filing date of the prior application and the national or PCT international filing date of this application:

| (App. Serial No.) | (Filing Date) | (Status -patented, pending, etc.) |
|---|---|---|

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

---

**Full name of sole or first inventor:** Gregg Evan Rothermel

Inventor's signature: _____ 10/28/99 (Date)

Residence: Albany, Oregon

Citizenship: U.S.A.

Post Office address: 1850 NW Crocker Lane
Albany, Oregon 97321

**Full name of second co-inventor:** Margaret Myers Burnett

Inventor's signature: _____ 10/28/99 (Date)

Residence: Corvallis, Oregon

Citizenship: U.S.A.

Post Office address: P.O. Box 699
Corvallis, Oregon 97339

**Full name of second co-inventor:** Lixin Li

Inventor's signature: _____ _____
(Date)

Residence: Corvallis, Oregon

Citizenship: U.S.A.

Post Office address: 440 NW 23rd Street
Corvallis, Oregon 97330

Oregon State University

Citizenship: U.S.A.

Post Office address: P.O. Box 699
Corvallis, Oregon 97339

**Full name of second co-inventor:** Lixin Li

Inventor's signature: _____ 11/02/1999 (Date)

Residence: Annandale, Virginia

Citizenship: People's Republic of China

Post Office address: 4706 Nathan Hale Drive, Apartment 101
Annandale, Virginia 22003

Sequence = 1 Length = 5

PATENT APPLICATION
Attorney's Do. No. 6954-2

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: Gregg Evan Rothermel, Margaret Myers Burnett and Lixin Li

Serial No.:

Filed:

For: METHODOLOGY FOR TESTING SPREADSHEETS

Box Patent Application
Assistant Commissioner for Patents
Washington, D.C. 20231

POWER OF ATTORNEY BY ASSIGNEE OF ENTIRE INTEREST
AND REVOCATION OF PRIOR POWERS

I, Wilson C. "Toby" Hayes, Vice President for Research, OREGON STATE UNIVERSITY, whose address is A312 Kerr Administration Building, Corvallis, Oregon 97331-2140, assignee of all right, title and interest of the above-described U.S. patent application, by the assignment submitted under separate cover for recordal (copy enclosed), represent that I am empowered to sign on behalf of assignee, OREGON STATE UNIVERSITY, as evidenced by the attached copy of the Confirmation of Agent's Authority in Connection with U.S. and Foreign Patent Applications.

As assignee of the above-identified application, all powers of attorney previously given are hereby revoked and the following attorneys and/or patent agents are hereby appointed to file a U.S. patent application based on this provisional application, to file a corresponding international application, to prosecute and transact all business in the Patent and Trademark Office connected therewith:

Customer No. 20575

| Attorney Name | Registration No. |
|---|---|
| Jerome S. Marger | 26,480 |
| Alexander C. Johnson, Jr. | 29,396 |
| Alan T. McCollom | 28,881 |
| James G. Stewart | 32,496 |

| | |
|---|---|
| Glenn C. Brown | 34,555 |
| Stephen S. Ford | 35,139 |
| Gregory T. Kavounas | 37,862 |
| Scott A. Schaffer | 38,610 |
| Joseph S. Makuch | 39,286 |
| James E. Harris | 40,013 |
| Graciela G. Cowger | 42,444 |
| Ariel Rogson | 43,054 |
| Craig R. Rogers | 43,888 |

Direct all telephone calls to Alexander C. Johnson, Jr. at (503) 222-3613 and send all correspondence to:

MARGER JOHNSON & McCOLLOM, P.C.
1030 S.W. Morrison Street
Portland, Oregon 97205

**OREGON STATE UNIVERSITY**

Date: 10/25/99

William Hostetler
Director of Technology Transfer

OSU 99-139